Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de algoritmos para la simulación mediante computación con membranas en dominios de aplicación biológicos

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Kevin Martínez García

Tutor: José María Sempere Luna

Cotutor: Marcelino Campos Francés

Curso 2020-2021

Resum

La computació amb membranes és un model de computació que s'inspira en el funcionament d'una cèl·lula eucariota animal. El model de sistemes P de transició proposat per Gheorghe Păun i els diferents tipus derivats d'aquest resulten de gran interès tant teòric com pràctic per les seves nombroses aplicacions en el món real. El nostre treball es centra en el desenvolupament de diferents algoritmes per als sistemes P estocàstics, un sistema que tracta de modelitzar les reaccions bioquímiques que tenen lloc en una cèl·lula mitjançant el càlcul de valors de propensidad d'aquestes reaccions. Els algoritmes desenvolupats s'apliquen a diferents escenaris i més concretament, a un escenari que simula l'expansió de virus COVID-19 en un entorn format per 500 unitats familiars. Els resultats de la simulació permeten extreure conclusions i observar l'evolució de virus en una població en la qual s'introdueix un infectat i es deixa evolucionar.

Paraules clau: membrana, algoritme, estocàstic, simulació, cèl·lula, COVID-19

Resumen

La computación con membranas es un modelo de computación que se inspira en el funcionamiento de una célula eucariota animal. El modelo de sistemas P de transición propuesto por Gheorghe Păun y los diferentes tipos derivados de este resultan de gran interés tanto teórico como práctico por sus numerosas aplicaciones en el mundo real. Nuestro trabajo se centra en el desarrollo de diferentes algoritmos para los sistemas P estocásticos, un sistema que trata de modelizar las reacciones bioquímicas que ocurren en una célula mediante el cálculo de valores de propensidad de dichas reacciones. Los algoritmos desarrollados se aplican a diferentes escenarios y más concretamente, a un escenario que simula la expansión del virus COVID-19 en un entorno formado por 500 unidades familiares. Los resultados de la simulación permiten extraer conclusiones y observar la evolución del virus en una población en la que se introduce un infectado y se deja evolucionar.

Palabras clave: membrana, algoritmo, estocástico, simulación, célula, COVID-19

Abstract

Membrane computing is a model of computation inspired by the functioning of an animal eukaryotic cell. The model of transitional P-systems proposed by Gheorghe Păun and the different types derived from it are of great theoretical and practical interest due to their numerous real-world applications. Our work focuses on the development of different algorithms for stochastic P-systems, a system that tries to model the biochemical reactions occurring in a cell by calculating propensity values of these reactions. The developed algorithms are applied to different scenarios and, more specifically, to a scenario that simulates the spread of the COVID-19 virus in an environment made up of 500 family units. The results of the simulation allow us to draw conclusions and observe the evolution of the virus in a population in which an infected person is introduced and allowed to evolve.

Key words: membrane, algorithm, stochastic, simulation, cell, COVID-19

Índice general

Índice general v
Índice de figuras vii
Índice de algoritmos viii

1 Introducción 1
1.1 Motivación . . . 1
1.2 Objetivos y problemas a resolver . . . 2
1.3 Estado del arte . . . 2
1.4 Estructura de la memoria . . . 3
2 Conceptos básicos de computación con membranas. Sistemas P de transición 5
2.1 Conceptos básicos . . . 5
2.2 Introducción a los multiconjuntos y estructuras de membranas . . . 9
2.2.1 Multiconjuntos . . . 9
2.2.2 Estructuras de membranas . . . 9
2.3 Súper células . . . 10
2.4 Sistemas P de transición . . . 12
3 Programación del simulador 17
3.1 Adaptación del modelo de sistemas P de transición a las necesidades experimentales . . . 17
3.2 Pasos previos. Configuración inicial y reglas . . . 18
3.3 Selección de reglas y ejecución . . . 20
3.4 Salida del sistema y recopilación de información . . . 21
4 Sistemas estocásticos basados en propensión. Algoritmos de ejecución de las reglas 23
4.1 Sistemas P estocásticos . . . 23
4.2 Esquema de selección de reglas. Bloques de prioridad . . . 25
4.3 Algoritmo 1 de aplicación . . . 26
4.4 Algoritmo 2 de aplicación . . . 32
5 Escenarios. Experimentación y resultados 35
5.1 Escenario 1 . . . 35
5.1.1 Escenario 1 - Algoritmo 1 . . . 36
5.1.2 Escenario 1 - Algoritmo 2 . . . 37
5.1.3 Comparación Algoritmo 1 - Algoritmo 2 . . . 38
5.2 Escenario 2 . . . 39
5.2.1 Escenario 2 - Algoritmo 1 . . . 40
5.2.2 Escenario 2 - Algoritmo 2 . . . 41
5.2.3 Comparación Algoritmo 1 - Algoritmo 2 . . . 42
5.3 Escenario 3 . . . 43
5.3.1 Escenario 3 - Algoritmo 1 . . . 44
5.3.2 Escenario 3 - Algoritmo 2 . . . 46
5.3.3 Comparación Algoritmo 1 - Algoritmo 2 . . . 48
5.4 Escenario 4 . . . 49

5.4.1 Escenario 4 - Algoritmo 1 50
5.4.2 Escenario 4 - Algoritmo 2 51
5.4.3 Comparación Algoritmo 1 - Algoritmo 2 52
5.5 Escenario COVID-19 52
5.5.1 Escenario COVID-19 - Algoritmo 1 56
6 Conclusiones **61**
Bibliografía **63**

Índice de figuras

2.1 Representación mediante diagramas de Venn de B como subconjunto de A. 6
2.2 Operaciones con los conjuntos A y B 7
2.3 Súper Célula de ejemplo. 11
2.4 Sistema P de ejemplo 1. 13
2.5 Sistema P de ejemplo 2. 14

3.1 Transformación de un sistema P a fichero xml de configuración 20

4.1 Sistema P de ejemplo para formación de bloques. 28
4.2 Ejemplo 1 para Algoritmo 1. 30
4.3 Ejemplo 2 para Algoritmo 1. 31
4.4 Ejemplo 1 para Algoritmo 2. 33

5.1 Escenario 1. 36
5.2 Progresión objetos Escenario 1 - Algoritmo 1 37
5.3 Ajuste por mínimos cuadrados Escenario 1 - Algoritmo 1. 37
5.4 Progresión objetos Escenario 1 - Algoritmo 2 38
5.5 Proporción por objeto para cada algoritmo 39
5.6 Escenario 2. 39
5.7 Progresión objetos Escenario 2 - Algoritmo 1. 40
5.8 Progresión objetos Escenario 2 - Algoritmo 2 41
5.9 Progresión c's Escenario 2 - Algoritmo 2. 42
5.10 Multiplicidad total de objetos para cada algoritmo 43
5.11 Escenario 3. 44
5.12 Progresión objetos Escenario 3 - Algoritmo 1. 45
5.13 Progresión a's Escenario 3 - Algoritmo 1. 46
5.14 Progresión objetos Escenario 3 - Algoritmo 1. 47
5.15 Escenario 4 49
5.16 Progresión objetos Escenario 4 - Algoritmo 1. 50
5.17 Progresión objetos Escenario 4 - Algoritmo 2. 51
5.18 Escenario COVID-19 54
5.19 Progresión sanos, infectados, inmunes y fallecidos escenario COVID-19. . 56
5.20 Progresión infectados por rol escenario COVID-19 57
5.21 Progresión fallecidos por rol escenario COVID-19 58
5.22 Progresión inmunizados por rol escenario COVID-19 58
5.23 Repuntes en las infecciones de trabajadores escenario COVID-19 59

Índice de algoritmos

4.1 Esquema general de ejecución 25
4.2 Algoritmo de obtención de bloques de reglas por prioridad 26
4.3 Algoritmo de obtención de bloques de reglas 27
4.4 Algoritmo 1 de aplicación 29
4.5 Algoritmo para la obtención de un número entero de ejecuciones 30
4.6 Algoritmo 2 de aplicación 32

CAPÍTULO 1
Introducción

La computación con membranas es un paradigma de computación inspirado en la estructura y el funcionamiento de organismos vivos [21] basado en la noción de estructura de membranas [14]. Una célula viva podría considerarse a nivel abstracto como un mecanismo que procesa información codificada bioquímicamente. En esencia, una reacción bioquímica toma ciertas sustancias conocidas como reactivos y genera otras sustancias potencialmente diferentes conocidas como producto de la reacción. Esto puede asemejarse a la forma en qué un algoritmo toma cierta información de entrada y la transforma en información de salida. La computación con membranas trata por tanto de imitar la forma en la que una célula realiza sus operaciones sin aspirar a plantear un modelo de célula viva.

Veremos en posteriores capítulos como procesos comunes a nivel celular como la endocitosis (transporte de cierta sustancia al interior de una región celular), la exocitosis (expulsión de cierta sustancia al exterior de una región celular) o ciertos procesos de división celular como la mitosis tienen su operación equivalente dentro del modelo de computación con membranas.

Mucho se ha escrito [7, 17] sobre las potenciales aplicaciones de la computación con membranas en contextos que aplican a problemas del mundo contemporáneo. En lo que respecta a nuestro trabajo, nos centraremos en dominios de aplicación biológicos e implementaremos dos algoritmos que permitirán simular la evolución de un total de cinco escenarios que expondremos en el capítulo 5. En cuanto a este capítulo, a continuación explicaremos brevemente la motivación de este proyecto seguido de los objetivos a cumplir y que problemas tratamos de resolver, examinaremos el estado del arte en esta materia y finalmente expondremos la estructura de la memoria.

1.1 Motivación

La motivación de este proyecto recae en el interés tanto teórico como práctico. Por una parte, se tenía interés en el estudio de los elementos teóricos de este paradigma de computación, así como en los diferentes algoritmos que se pueden implementar en base al mismo. En lo que respecta al interés práctico, este surgía de la implementación de estos algoritmos y la experimentación con diferentes escenarios para observar el comportamiento de los mismos. Además, resultaba de especial interés la experimentación y extracción de conclusiones con un escenario que simulaba la expansión del virus COVID-19 en una población donde existen ciertos grupos que llevan a cabo sus rutinas diarias. En la actualidad, la existencia del virus nos ha obligado a cambiar nuestra forma de proceder en todos los ámbitos de la vida diaria. Elementos tan comunes en nuestro día a día como las reuniones con amistades o compañeros de trabajo se deben realizar ahora

con mucha más precaución y requieren de mayor cautela por parte de todas las partes involucradas.

Por esto mismo, consideramos que estas simulaciones que permiten estudiar el desarrollo y la expansión del virus pueden resultar extremadamente útiles a la hora de hacer predicciones e incluso plantear medidas preventivas y paliativas de los potenciales efectos que puede tener sobre la población.

1.2 Objetivos y problemas a resolver

Los principales objetivos a cumplir durante el desarrollo del proyecto fueron los que se enumeran a continuación.

- Aprender y familiarizarse con los conceptos básicos de la computación con membranas.
- Adaptar algoritmos de simulación estocásticos y probabilistas clásicos en la computación con membranas tales como el algoritmo de Gillespie, los algoritmos de dinámica de poblaciones etc.
- Realizar un estudio teórico de computación con membranas y de algunos algoritmos de aplicación de reglas con el propósito final de lograr simular un escenario de interés biológico como escenarios de expansión del virus COVID-19.

1.3 Estado del arte

La computación con membranas fue introducida por Gheorghe Păun en 1999 [14]. Desde entonces se han llevado a cabo numerosas investigaciones en esta rama de la informática. En la actualidad se han celebrado algunos "talleres" de computación con membranas dónde se presentan numerosos estudios y aplicaciones en este área de investigación. Podemos destacar por ejemplo el *8th International Workshop on Membrane Computing* [4] dónde se presentaron proyectos tan interesantes como los sistemas Q-UREM [9] que suponen un intento de introducción de técnicas y nociones derivadas de la mecánica cuántica en la computación con membranas. Otro proyecto interesante podría ser el diseño de sistemas P con reglas de comunicación por petición [3] es decir, sistemas P donde es posible la comunicación de objetos entre membranas mediante la ocurrencia de ciertos símbolos *query* en las cadenas.

Destaca también otro proyecto de gran interés teórico que parte de un *spiking neural P system* que resuelve el problema de la suma de los subconjuntos en un número constante de pasos y estudia como afectan las propiedades de estos sistemas a las capacidades de resolver problemas numéricos NP-completos [10]. Existen además proyectos tan prometedores como el *P-Lingua* un lenguaje de programación para la computación con membranas que busca convertirse en el estándar para definir sistemas P [22]. El lenguaje ha sido desarrollado por miembros del *Research Group on Natural Computing* de la Universidad de Sevilla. *P-Lingua* da soporte a diversos modelos de computación con membranas (como los sistemas P de transición [14], los sistemas P con membranas activas [15], sistemas P *symport/antiport* [16] y los sistemas P probabilísticos [13] entre otros) y dispone de varios simuladores para dar soporte a estos modelos.

En definitiva, con estos proyectos que hemos expuesto, queremos destacar que en la actualidad se sigue desarrollando un proceso de investigación notable en esta área y existe un interés tanto teórico como práctico en este campo de la informática.

1.4 Estructura de la memoria

En los capítulos que siguen, vamos a hacer una exposición de todo el trabajo que se desarrolló durante este proyecto. En primer lugar, comenzaremos haciendo una introducción teórica a la computación con membranas y los sistemas P de transición, tratando de introducir los conceptos necesarios para su comprensión y el modelo que utilizamos durante nuestras simulaciones. A continuación, describiremos brevemente el proceso de diseño e implementación del simulador así como ciertos cambios que introdujimos en el modelo de sistemas P de transición para adaptarlos a nuestras necesidades de experimentación. Seguidamente, haremos una breve introducción de los sistemas P estocásticos basados en propensión junto con una descripción de los algoritmos de selección de reglas implementados. Terminaremos la memoria con una exposición detallada de los escenarios con los que se condujeron los experimentos y los resultados obtenidos para cada caso y por último, extraeremos las conclusiones de todo el trabajo realizado.

CAPÍTULO 2

Conceptos básicos de computación con membranas. Sistemas P de transición

La computación con membranas se puede clasificar dentro de lo que se conoce como computación natural o bioinspirada, es decir, trata de simular el modo en el que la naturaleza actúa sobre la materia [21]. Existen aportaciones significativas a este campo, como podrían ser las redes neuronales o los algoritmos genéticos. En definitiva, lo que se busca mediante este paradigma de computación es implementar ciertos mecanismos de la naturaleza sobre medios de silicio.

En lo que respecta al modelo que vamos a definir, tomaremos el concepto de membrana en el sentido más amplio del término, es decir, podemos considerar que una membrana es una entidad que separa el interior y el exterior de una célula o una entidad más abstracta que se utiliza para separar regiones dentro de un ecosistema. Para poder comprender con mayor claridad en qué consiste la computación con membranas, introduciremos en primer lugar una serie de conceptos básicos de la teoría de conjuntos y la teoría de lenguajes formales así cómo dos conceptos esenciales que son, el concepto de multiconjunto y el de estructura de membranas.

Seguidamente, describiremos el concepto de súper célula, es decir, una estructura de membranas que puede contener una serie de entidades abstractas denominadas objetos. Finalmente, pasaremos a considerar estructuras de membranas con objetos que pueden evolucionar de acuerdo a lo que se conoce como reglas de evolución, definiendo de esta forma los sistemas P de transición.

Además, resulta especialmente interesante desde un punto de vista teórico mencionar que los sistemas P de transición son capaces de caracterizar el conjunto recursivamente enumerable de los números naturales [14], es decir, proporcionan completitud computacional. Además, únicamente necesitamos dos membranas para conseguir esta caracterización. Esto nos permite afirmar que los sistemas P de transición son equivalentes a una máquina de Turing, lo que es interesante desde la perspectiva de la computabilidad y la complejidad.

2.1 Conceptos básicos

Antes de proceder con las definiciones que respectan a los sistemas P, vamos a introducir algunos conceptos de teoría de conjuntos y teoría de lenguajes formales de los que nos serviremos más adelante. Los conceptos que se presentan a continuación han obte-

nidos en [20] [8]. Empecemos definiendo el concepto de conjunto. Un conjunto es una colección no ordenada de objetos, denominados elementos o miembros del conjunto. Se dice que un conjunto contiene a sus elementos. Escribimos $a \in A$ para denotar que a es un elemento del conjunto A. La notación $a \notin A$ denota que a no es un elemento del conjunto A.

La notación habitual que emplearemos denota los conjuntos mediante letras mayúsculas y a los elementos del conjunto mediante letras minúsculas. Existen diversas formas de definir un conjunto, siendo las más habituales las dos que exponemos a continuación.

- **Por extensión**: en este tipo de notación todos los elementos del conjunto aparecen entre llaves y separados por comas. Por ejemplo, $A = \{a, b, c, d\}$ denotaría al conjunto A que contiene los elementos a, b, c y d.
- **Por comprensión**: mediante este tipo de notación caracterizamos todos los elementos del conjunto enunciando la propiedad o propiedades que deben poseer para formar parte del mismo. Por ejemplo, $A = \{x \in \mathbb{N} \mid 1 \leq x \leq 10\}$, sería la definición por comprensión del conjunto que contiene los 10 primeros números naturales.

Habitualmente nos encontraremos en situaciones en las los elementos de un conjunto son también los elementos de otro conjunto diferente. Decimos que A es un subconjunto de B si y sólo sí todos los elementos de A son también elementos de B [20]. Utilizamos la notación $A \subseteq B$ para denotar que A es un subconjunto de B. Una forma de visualizar un conjunto y sus posibles subconjuntos es mediante diagramas de Venn. Los diagramas de Venn fueron introducidos por el matemático John Venn en 1881 y se utilizan para representar los conjuntos y sus subconjuntos mediante figuras geométricas como círculos o rectángulos. Un ejemplo del uso de diagramas de Venn para representar conjuntos y sus subconjuntos es el que aparece en la Figura 2.1.

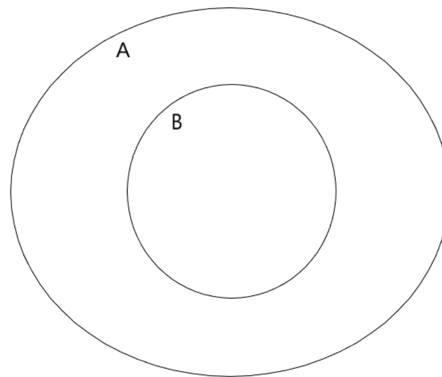


Figura 2.1: Representación mediante diagramas de Venn de B como subconjunto de A.

Otro aspecto a considerar son las operaciones que pueden llevarse a cabo entre conjuntos. Una de estas operaciones es la unión de conjuntos. Sean A y B dos conjuntos, la unión de ambos conjuntos denotada como $A \cup B$, es el conjunto que contiene los elemenotos que están o bien en A, o bien en B o bien en ambos. En notación por comprensión, podemos definir el conjunto $A \cup B$ como sigue

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

Otra operación a tener en cuenta es la intersección de conjuntos. Sean A y B dos conjuntos, la intersección de A y B denotada cómo $A \cap B$ es el conjunto que contiene a

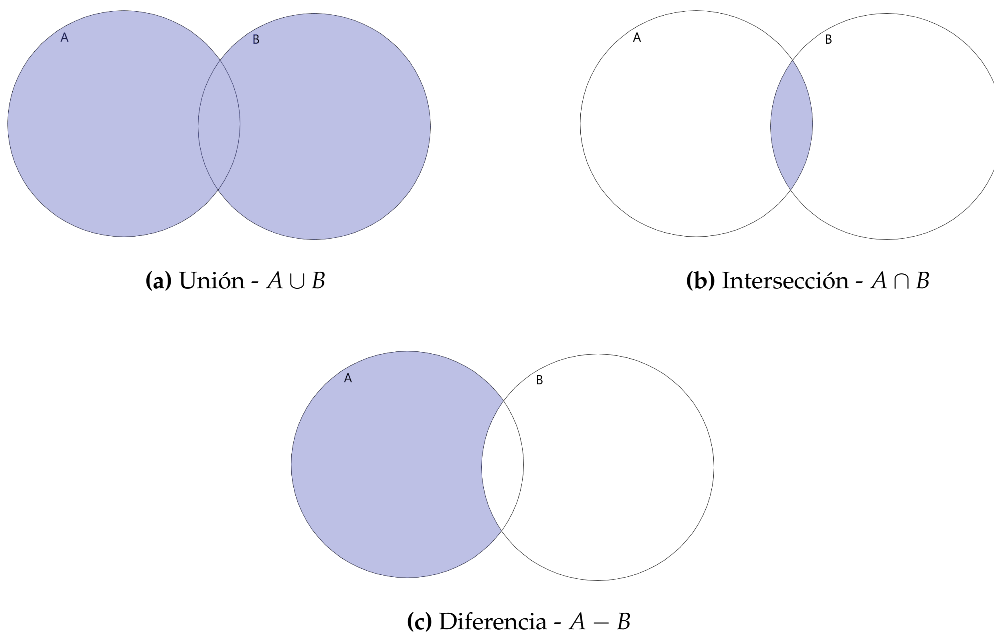
los elementos que están tanto en A cómo en B. En notación por comprensión, podemos definir el conjunto $A \cap B$ como sigue

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

Por último, pasamos a considerar la diferencia de conjuntos. Sean A y B dos conjuntos, la diferencia de A y B denotada cómo $A - B$ es el conjunto que contiene los elementos que están en A pero no están en B. En notación por comprensión, podemos definir el conjunto $A - B$ como sigue

$$A - B = \{x \mid x \in A \wedge x \notin B\}$$

Una representación mediante diagramas de Venn de estas operaciones, es la que aparece en la Figura 2.2 a continuación. En estas figuras, las secciones en azul representan el conjunto resultante de cada operación.



(a) Unión - $A \cup B$

(b) Intersección - $A \cap B$

(c) Diferencia - $A - B$

Figura 2.2: Operaciones con los conjuntos A y B

Una vez introducidos estos conceptos de la teoría de conjuntos que consideramos importantes, pasamos a exponer algunos conceptos de la teoría de lenguajes formales. Empezamos con la definición de alfabeto, que es un conjunto finito de símbolos. El símbolo es un primitivo de la teoría de lenguajes formales y se suelen utilizar letras del alfabeto latino o dígitos para representarlos. Algunos ejemplos de alfabetos son los que siguen.

$$\Sigma = \{a, b, c\} \quad \Gamma = \{0, 1\} \quad \Pi = \{\triangle, \bigtriangledown, \bigcirc\}$$

Por otra parte, una cadena es una secuencia finita y ordenada de símbolos de un alfabeto [8]. Por ejemplo, $x = aabbcc$ sería una cadena formada con los símbolos del alfabeto Σ y $y = 0101$ sería una cadena formada por los símbolos del alfabeto Γ. Habitualmente estas cadenas se denotan mediante las últimas letras del alfabeto latino. Denotaremos la cadena vacía con el símbolo λ. Una propiedad interesante de las cadenas es la longitud de las mismas. La longitud de una cadena es el número de símbolos que contiene y se

denota cómo $|x|$ siendo x una cadena sobre cierto alfabeto. Formalmente, si x e y son palabras sobre cierto alfabeto Σ y $a \in \Sigma$ entonces

$$|x| = \begin{cases} 0 & \text{si x = } \lambda \\ 1 + |y| & \text{si x = ay} \end{cases}$$

El número de veces que cierto símbolo a aparece en una cadena x se denota cómo $|x|_a$. Para cierto alfabeto Σ definimos Σ^n como el conjunto de las palabras de longitud n sobre dicho alfabeto. A partir de esta definición se construye $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$ conocida como la clausura transitiva o de Kleene. En otras palabras Σ^* es el conjunto de todas las posibles cadenas sobre Σ. A partir de esta última definición se deriva el concepto de lenguaje. Un lenguaje es un subconjunto de Σ^* es decir, un conjunto finito o infinito de cadenas. Decimos que un lenguaje es finito si contiene un número finito de cadenas y en caso contrario es infinito numerable.

Para terminar con esta sección, vamos a introducir el concepto de relación binaria. Sean A y B dos conjuntos, una relación binaria entre A y B es un subconjunto de $A \times B$, es decir, del producto cartesiano entre A y B. En otras palabras, una relación binaria entre A y B es un conjunto R de parejas ordenadas dónde el primer elemento pertenece a A y el segundo a B. Utilizamos la notación aRb para denotar que $(a, b) \in R$ y $a \not R b$ para denotar que $(a, b) \notin R$. Mencionar que $A \times B$ es el producto cartesiano de A con B, es decir, el conjunto de todas las parejas ordenadas (a, b) dónde $a \in A$ y $b \in B$. Formalmente,

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$$

Por otra parte, una relación en un conjunto A es una relación de A con él mismo. Este tipo de relaciones pueden clasificarse en función de ciertas propiedades que exponemos a continuación.

- **Reflexiva**: una relación R en un conjunto A es reflexiva si $(a, a) \in R$ para todo elemento $a \in A$.
- **Simétrica**: una relación R en un conjunto A es simétrica si $(b, a) \in R$ siempre que $(a, b) \in R$ para todo $a, b \in A$.
- **Antisimétrica**: una relación R en un conjunto A es antisimétrica si para todo $a, b \in A$, si $(a, b) \in R$ y $(b, a) \in R$ entonces $a = b$.
- **Transitiva**: una relación R en un conjunto A es transitiva si siempre que $(a, b) \in R$ y $(b, c) \in R$ entonces $(a, c) \in R$.

Una relación R en un conjunto A decimos que es de equivalencia si es reflexiva, simétrica y transitiva. Dos elementos a y b relacionados mediante una relación de equivalencia decimos que son equivalentes. Habitualmente, se utiliza la notación $a \backsim b$ para denotar que a y b son equivalentes con respecto a cierta relación de equivalencia. Sea R una relación de equivalencia en un conjunto A. El conjunto de todos los elementos que están relacionados con un elemento $a \in A$ se denomina clase de equivalencia de a. La clase de equivalencia de a con respecto a R se denota como $[a]_R$. Formalmente, estas clases de equivalencia se definen como sigue

$$[a]_R = \{s \mid (a, s) \in R\}$$

Al conjunto formado por las clases de equivalencia de la relación R, se le denomina conjunto cociente y se denota como A/R y se define como sigue

$$A/R = \{[a]_R \mid a \in A\}$$

2.2 Introducción a los multiconjuntos y estructuras de membranas

2.2.1. Multiconjuntos

En cualquier sistema P que vayamos a considerar aparece la posibilidad de que un objeto esté presente más de una sola vez en cierta región delimitada por una membrana. Para abstraer y denotar esta realidad utilizamos el concepto de multiconjunto que pasamos a definir. Sea U un conjunto arbitrario cualquiera, un multiconjunto es un par (U, M) dónde $M : U \to \mathbb{N}$ es una correspondencia desde U a los números naturales. Para cada $a \in U$, la multiplicidad de a se denota cómo $M(a)$. Es habitual ver escrita la correspondencia M como un conjunto de pares ordenados $\{(a, M(a)) \mid a \in U\}$.

Es importante también el concepto de soporte de un multiconjunto que se puede definir como *supp(M)* = $\{a \in U \mid M(a) > 0\}$. Sean M_1, M_2 dos multiconjuntos. Por una parte, decimos que M_1 está incluido en M_2 si y sólo si $M_1(a) \leq M_2(a)$, para todo $a \in supp(M_1)$. La unión de M_1 con M_2 es el multiconjunto $M_1 \cup M_2 : U \to \mathbb{N}$ definido cómo ($M_1 \cup M_2$)(a) = $M_1(a) + M_2(a)$ para todo $a \in U$. Por último, la diferéncia $M_1 - M_2$ es el multiconjunto $M_1 - M_2 : U \to \mathbb{N}$ dado por ($M_1 - M_2$)(a) = $M_1(a) - M_2(a)$ para todo $a \in U$.

Habitualmente, los multiconjuntos M con soporte finito $\{(M(a_1), a_1), (M(a_2), a_2), ..., (M(a_n), a_n)\}$ se denotarán como la cadena $a_1^{M(a_1)} a_2^{M(a_2)} ... a_n^{M(a_n)}$ o cualquier permutación de la misma. Este tipo de notación será la que nosotros utilizaremos para representar los objetos contenidos en cierta región delimitada por una membrana. En la sección 2.1 vimos cómo una cadena era una secuencia finita de símbolos sobre cierto alfabeto. De esta definición se desprende que cualquier cadena $x \in V^*$ define un multiconjunto sobre los símbolos de V que se puede denotar como $m(x) = \{(a, |x|_a) \mid a \in V\}$. Por ejemplo, la cadena $x = aabbc$ (o cualquier permutación de la misma) definiría el multiconjunto $m(x) = \{(a, 2), (b, 2), (c, 1)\}$.

2.2.2. Estructuras de membranas

Como introdujimos al principio de este mismo capítulo, una membrana puede utilizarse como una entidad delimitadora en diferentes contextos. Podríamos, por tanto, utilizar una membrana para delimitar las diferentes regiones de una célula eucariota animal o para separar las diferentes secciones de un hospital que estamos simulando mediante un computador.

Por tanto, en los sistemas que vamos a estudiar tendremos una disposición de las membranas en las que unas estarán contenidas dentro de otras o serán adyacentes las unas con las otras (utilizaremos el término de membranas vecinas). Esta forma de disponer las membranas requiere de una abstracción matemática.

Consideremos el lenguaje MS sobre el alfabeto $\{[,]\}$ cuyas cadenas se definen de forma recursiva como sigue:

1. $[] \in MS$.

2. Si $\mu_1, ..., \mu_n \in MS$, $n \geq 1$, entonces $[\mu_1 ... \mu_n] \in MS$.
3. No hay nada más en MS.

Si consideramos la relación $\backsim$ sobre MS: $x \backsim y$ si y sólo si podemos escribir dos cadenas con la forma $x = \mu_1...\mu_i\mu_{i+1}...\mu_n, y = \mu_1...\mu_{i+1}\mu_i...\mu_n$ con $\mu_j \in MS, 1 \leq j \leq n, n > 0$, es decir dos pares de corchetes al mismo nivel se intercambian así como sus contenidos. Esta relación es de equivalencia por lo que denotamos como $\overline{MS}$ el conjunto de las clases de equivalencia de MS con respecto a la relación $\backsim$. Los elementos de $\overline{MS}$ son estructuras de membranas.

Dos pares de corchetes [,] denotarían una membrana en una estructura de membranas. Sea μ una estructura de membranas, el número de membranas que contiene es el grado de μ y se denota como $deg(\mu)$. La membrana más externa de una estructura es la membrana piel de μ. Una membrana que no contiene más membranas en su interior, es decir, una membrana con la forma [ ] es lo que se conoce como una membrana elemental. La profundidad de una estructura μ se denota como $dep(\mu)$ y se define recursivamente como sigue:

1. Si μ = [ ], entonces $dep(\mu) = 1$
2. Si $\mu = [\mu_1...\mu_n]$ para algún $\mu_1, ..., \mu_n \in MS$ entonces $dep(\mu) = max\{dep(\mu_i)|1 \leq i \leq n\} + 1$

Más allá de estas definiciones abstractas en muchas ocasiones resultará especialmente útil tener una representación visual de las estructuras de membranas. Esto permite identificar rápidamente cómo están dispuestas las membranas así como los objetos que les corresponden a cada una. Para esta representación visual utilizaremos diagramas de Venn. Esta representación visual ayuda a dar sentido a la relación de equivalencia $\backsim$ pues parece evidente que el orden en que se disponen membranas vecinas es irrelevante y únicamente nos interesa la topología del sistema, es decir, cómo se disponen las membranas las unas dentro de las otras.

Mencionar antes de proseguir que, es habitual encontrar estructuras de membranas en las que los corchetes aparecen con un subíndice. Esto se utiliza para identificar las membranas y veremos que existe una definición formal de esta notación. Es importante recalcar que esta identificación puede no ser única, es decir, podemos tener membranas distintas pero con el mismo identificador.

2.3 Súper células

Antes de enfrentarnos a la definición de los sistemas P de transición, conviene ampliar los conceptos vistos hasta el momento mediante la adición de objetos a las estructuras de membranas. Para ello, consideremos una estructura de membranas μ de grado $deg(\mu) = n$ y establezcamos una correspondencia uno a uno entre las membranas y sus etiquetas. Por simplicidad, consideremos que esta correspondencia es de cada membrana con los naturales entre 1 y n.

Consideremos además un conjunto U que contiene objetos. Si cada región delimitada por la membrana de la estructura μ contiene un multiconjunto $M_i : U \to \mathbb{N}$ para $1 \leq i \leq n$ decimos que tenemos una súper célula. A cada uno de estos multiconjuntos que se corresponden a cada región se les conoce como los contenidos de la región.

En cuanto a la representación visual, al igual que en el apartado anterior se utilizan diagramas de Venn. En este caso, se representan además los multiconjuntos que contiene

cada región. Para consolidar todo lo visto hasta el momento, podemos partir de la Figura 2.3 que aparece a continuación.

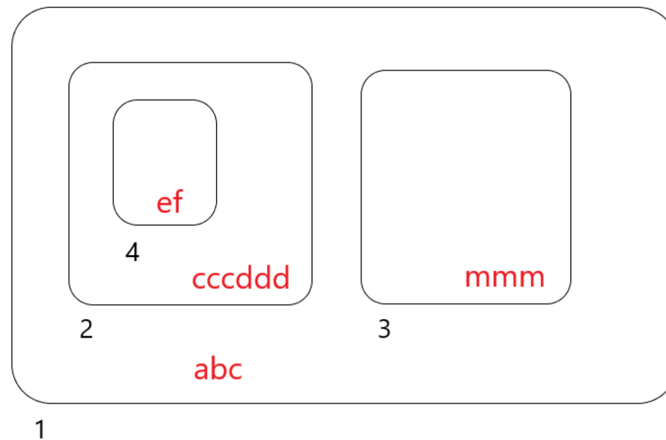


Figura 2.3: Súper Célula de ejemplo.

En esta figura podemos ver como tenemos un total de cuatro membranas (numeradas correspondientemente del uno al cuatro). Por una parte, la estructura de membranas podría representarse mediante la siguiente cadena $\mu = [_1[_2[_4]_4]_2[_3]_3]_1$. De esta estructura mencionar que, como indicamos en el subapartado 2.2.2, hemos añadido un subíndice a cada par de corchetes para indicar la etiqueta que le corresponde a cada uno.

Por otra parte, tenemos cuatro multiconjuntos, uno para cada región que pueden definirse como sigue.

- $M_1 = \{(1,a),(1,b),(1,c)\}$
- $M_2 = \{(3,c),(3,d)\}$
- $M_3 = \{(3,m)\}$
- $M_4 = \{(1,e),(1,f)\}$

Podemos ver como, al representar los contenidos de las membranas en la Figura 2.3, hemos utilizado la notación en forma de cadena que explicábamos en el apartado 2.2.1. Recalcar de nuevo que, cualquier permutación de estas cadenas sirve para representar el mismo multiconjunto, es decir, las cadenas *cccddd* y *cdcdcd* son equivalentes.

Otro aspecto de interés a comentar es que el grado de esta súper célula es $deg(\mu) = 4$ y la profundidad es $dep(\mu) = 3$. Por otra parte, cada membrana de esta súper célula concuerda a su vez con la definición de súper célula. En otras palabras, cada membrana constituye a su vez una súper célula. Con todo los visto hasta ahora ya podemos proceder a definir los sistemas P de transición que, en esencia, son los sistemas con los que vamos a trabajar durante las simulaciones y experimentos.

2.4 Sistemas P de transición

Para concluir con este capítulo, procederemos a introducir los sistemas P de transición. El último paso de esta construcción que hemos expuesto hasta el momento consiste en permitir la evolución de los objetos de una súper célula mediante la adición de las denominadas reglas de transición o evolución. Por tanto, un sistema P de transición de grado n para $n \geq 1$ es una tupla que adopta la siguiente forma.

$$\Pi = (V, \mu, w_1, ..., w_n, (R_1, \rho_1), ..., (R_n, \rho_n), i_0)$$

dónde:

1. V es un alfabeto cuyos elementos son objetos.
2. μ es una estructura de membranas de grado n con las membranas etiquetadas en una correspondencia uno a uno con los elementos de cierto conjunto Λ. Habitualmente se numerarán con enteros de 1 a n.
3. $w_i, 1 \leq i \leq n$ son cadenas sobre V^*, es decir multiconjuntos sobre V asociados a las n regiones de μ.
4. $R_i, 1 \leq i \leq n$ son conjuntos finitos de reglas de evolución sobre los objetos de V asociadas a las regiones $1, 2, ..., n$ de μ.
5. $\rho_i, 1 \leq i \leq n$ son relaciones de orden parcial sobre R_i y determinan la prioridad entre las reglas de estos conjuntos.
6. i_0 es la membrana que constituye la salida de Π. En caso de que el sistema nunca detenga su ejecución, entonces la salida se observaría en su exterior.

De estas definiciones previas, resulta importante destacar ciertos conceptos. Por una parte, una regla de evolución es un par (u, v) que habitualmente se escribe como $u \rightarrow v$ donde u es una cadena sobre V y $v = v^{'}$ o $v = v^{'}\delta$ dónde $v^{'}$ es una cadena sobre:

$$(V \times \{here, out\}) \cup (V \times \{in_j | 1 \leq j \leq n\})$$

y δ es un símbolo especial que no pertenece a V. Es importante entender el significado de estas reglas para comprender cómo se aplican y cómo hacen evolucionar los objetos de una región. Para ello, vamos a partir del sistema P de transición que aparece en la Figura 2.4 y examinaremos las reglas que contiene.

En este caso tenemos un sistema P de transición con dos membranas y el multiconjunto $M_1 = \{(1, a)\}$ en la membrana con etiqueta 1. Esta membrana contiene dos reglas, por una parte tenemos la regla $a \rightarrow (b, here)$, esta regla nos indica que, cuando tengamos una a en la membrana 1 evolucionará para convertirse en una b que quedará también en la membrana 1. El elemento *here* es una forma de indicar que el objeto queda en la misma región. En cuanto a la regla $b \rightarrow (b, in_2)$, esta regla nos indica que cuándo tengamos una b en la región 1 evolucionará para convertirse en una b que entrará en la región con etiqueta 2 (siempre que esta región exista y esté contenida en la membrana 1). Por tanto, el elemento in_j nos indica que el objeto pasará a la región con etiqueta j.

Por último, en la membrana 2 tenemos la regla $b \rightarrow (c, out)$, esta regla nos indica que cuándo tengamos una b en la membrana 2 evolucionará para convertirse en una c que saldrá de esta membrana y pasará a formar parte de aquella que la contenga, en este

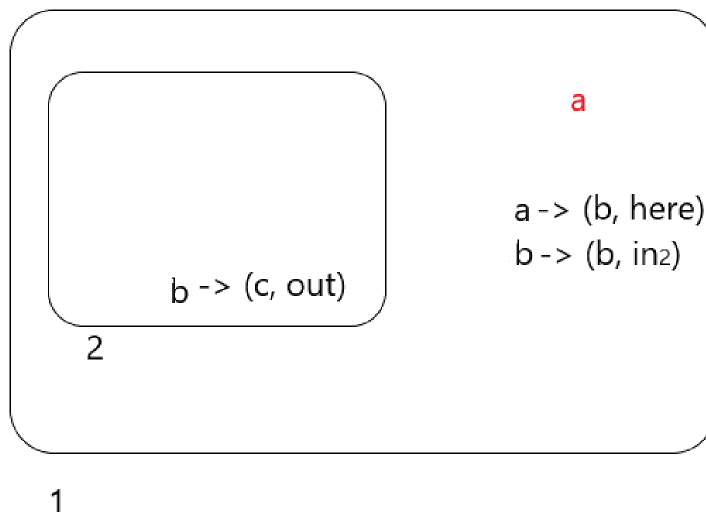


Figura 2.4: Sistema P de ejemplo 1.

caso, la membrana 1. Por tanto el elemento *out* indica que el objeto sale al exterior de la membrana que contiene dicha regla. En caso de que tengamos una regla con *out* en la membrana más externa del sistema P, entonces, el objeto saldrá al exterior del sistema y se dejará de tener en cuenta en las siguientes computaciones.

Un aspecto a tener en cuenta es que, habitualmente el elemento *here* se omite y, cuando un objeto no aparece con una "pareja" se entiende que nos estamos refiriendo a que los objetos quedan en la misma región donde se encuentra la regla. En los futuros ejemplo que propongamos omitiremos *here* para mayor brevedad.

Por otra parte, las relaciones de orden parcial ρ_i establecen un orden de prioridad a la hora de aplicar las reglas en cierta membrana, de nuevo, partiremos de un ejemplo muy similar al anterior que puede verse en la Figura 2.5. En este caso podemos ver como en la membrana 2 tenemos dos reglas que compiten por cierto objeto b. Sin embargo la relación de prioridad ρ_1 establecería que la regla $b \to (b, in_2)$ es más prioritaria que la regla $b \to (b, out)$, es decir, en caso de tener un objeto b en esta membrana, aplicaríamos siempre la regla $b \to (b, in_2)$. Destacar además cómo, en este ejemplo, hemos obviado el elemento *here* y entendemos que, en su ausencia, los objetos generados quedan en la membrana que contiene dichas reglas. Mencionar por último que es habitual también utilizar un tipo de notación dónde los *targets out* y in_j aparecen como subíndices de los objetos a los que afectan. Así, la última regla que hemos examinado se podría escribir también como $b \to b_{in_2}$. Utilizaremos de forma indistinta ambas notaciones pues son equivalentes.

Otro aspecto de las reglas de transición a tener en cuenta es la aparición del símbolo δ en su parte derecha. Este símbolo índica la disolución de una membrana, es decir, la membrana desaparece de la estructura de membranas y todos sus contenidos se trasladan a la membrana que la contenía. Una limitación a este tipo de reglas es que la piel del sistema P no puede disolverse, es decir, la membrana más externa del sistema no puede desaparecer.

Una última definición con respecto a las reglas que nos resultará útil para la agrupación de las mismas en bloques, es la definición formal de parte izquierda de la regla. Si bien previamente hemos introducido el concepto de parte izquierda de una regla, consideramos conveniente definirlo formalmente y especificar la notación que vamos a utilizar para referirnos a ella. En el capítulo 4, cuando estudiemos los diferentes algoritmos

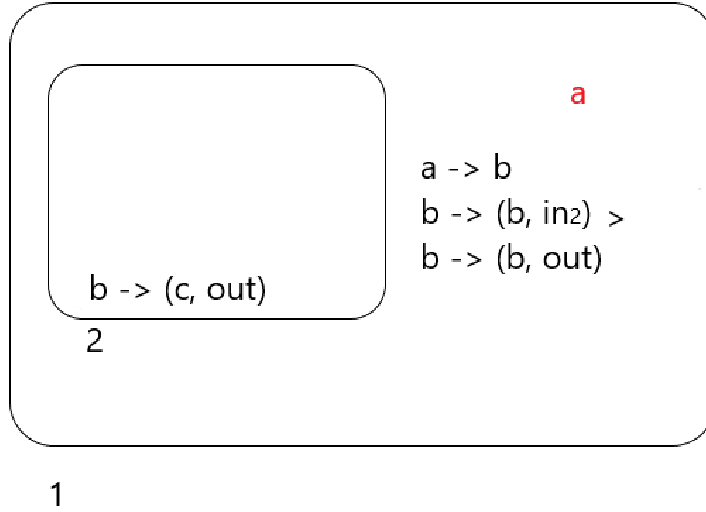


Figura 2.5: Sistema P de ejemplo 2.

de aplicación de reglas, veremos como será necesario agruparlas cuando compartan elementos de su parte izquierda. Por tanto, podemos definir la parte izquierda de una regla $r_j \in R_i, \forall R_i \in \Pi$, como un multiconjunto de la forma $LHS(r_j) = w$ siendo w una cadena sobre V^* y $r_j : w \rightarrow \alpha$.

También es vital tener en cuenta la maximalidad de los sistemas P de transición. Esta propiedad se refiere a que todos los objetos que puedan evolucionar bajo la aplicación de cierta regla de transición deben evolucionar. Es decir, no puede quedar ningún objeto en ninguna membrana que pudiendo evolucionar quede sin hacerlo. Además, las evoluciones de los objetos ocurren en paralelo en todas las membranas.

Habitualmente utilizaremos el término paso de computación para referirnos a la evolución maximal de todos los objetos de un sistema P. En otras palabras la evolución de los objetos del sistema hasta que no queda ninguna regla aplicable constituiría un paso de computación. En términos formales diríamos que cualquier secuencia μ', w'_{i_1}, ..., $w'_{i_k}, (R_{i_1}, \rho_{i_1})$, ...,$(R_{i_n}, \rho_{i_n})$ siendo μ' una estructura de membranas obtenida a partir de μ mediante la eliminación (disolución) de todas las membranas diferentes de i_1,..., i_n, siendo w'_{i_1}, ..., w'_{i_k} multiconjuntos sobre V y $\{i_1, ..., i_k\} \subseteq \{1, 2, ..., n\}$, es una configuración de Π. Por tanto, en un paso de computación el sistema parte de una configuración como la que sigue:

$$C_1 = \left(\mu', w'_{i_1}, ..., w'_{i_k}, (R_{i_1}, \rho_{i_1}), ..., (R_{i_k}, \rho_{i_k})\right)$$

y utilizando las reglas de transición $R_{i_1}, ..., R_{i_k}$ transiciona a otra configuración:

$$C_2 = \left(\mu'', w''_{j_1}, ..., w''_{j_l}, (R_{j_1}, \rho_{j_1}), ..., (R_{j_l}, \rho_{j_l})\right)$$

lo que supondría un paso de computación. Destacar que el cambio de subíndices denota la posibilidad de que ciertas membranas se disuelvan lo que cambiaría el grado del sistema. Con todo lo visto hasta el momento, podemos tener una noción de qué es un sistema P así como entender su funcionamiento a nivel operativo. Este modelo de sistema P de transición que se ha expuesto constituye un modelo básico [14] del que se han derivado modelos más complejos y potentes. Entre ellos, podemos destacar los sistemas P de

reescritura, los sistemas P con catalizadores [14], los sistemas P con membranas activas [15], los sistemas P probabilísticos [12] y los sistemas P estocásticos [21].

Los sistemas P con membranas activas son un tipo de sistemas P que contemplan reglas que actúan tanto sobre objetos como membranas. Es decir, este tipo de sistemas contempla la división, creación e incluso movimiento de membranas. En lo que respecta al alcance de este proyecto, nos centraremos únicamente en reglas que afecten a objetos y no tendremos en cuenta reglas que actúen directamente sobre membranas. Por otra parte, sí que se dará soporte a los sistemas P estocásticos. Sin embargo, por la complejidad de estos sistemas y la necesidad de proporcionar una explicación profunda de ciertos aspectos, consideramos adecuado dedicarles el capítulo 4 de esta memoria.

CAPÍTULO 3

Programación del simulador

Procedemos en este capítulo a hacer una exposición de cómo se desarrolló el simulador de los sistemas P de transición. En primer lugar, el simulador se desarrolló utilizando el lenguaje de programación *Java* en el entorno de desarrollo *BlueJ*. Por una parte, elegimos *Java* por ser uno de los lenguajes de uso común y orientado a objetos, lo que facilita la abstracción de las entidades más importantes de un sistema P en clases. Además, los simuladores desarrollados tanto en el proyecto P-Lingua [22], como para los proyectos *LOIMOS* y *ARES* [2] utilizaron este lenguaje, lo que facilitaba notablemente la asistencia en caso de que se necesitase ayuda con algún aspecto de la programación. Por otra parte, se eligió el entorno *BlueJ* por ser uno de los entornos de desarrollo más utilizados durante el transcurso de los estudios en Ingeniería Informática y contar con cierta experiencia en su manejo.

A continuación, pasaremos a explicar algunas de las adaptaciones que se hicieron al modelo de sistemas P presentado en el capítulo 2 para adecuarse a las necesidades de los experimentos propuestos como objetivo de este proyecto. Seguidamente, se expondrán algunos de los requisitos y procedimientos previos a lanzar la ejecución del sistema y, por último, explicaremos cómo funciona el proceso de selección y ejecución de reglas así como la obtención e interpretación de la salida.

3.1 Adaptación del modelo de sistemas P de transición a las necesidades experimentales

Como se ha explicado al inicio de este mismo capítulo, fue necesario modificar algunos aspectos del modelo enunciado en el capítulo 2 con el objetivo de adaptarlo a las necesidades de los experimentos y la simulación. Vamos a pasar a explicar los cambios y los motivos por los que se llevaron a cabo.

Por una parte, vimos que un sistema P de transición estaba compuesto por una estructura de membranas μ cuyas membranas estaban etiquetadas en una correspondencia uno a uno con los elementos de cierto conjunto Λ. Formalmente, podríamos decir que existe una correspondencia biyectiva entre el conjunto de las membranas y el conjunto Λ de los identificadores. Sin embargo, en nuestras simulaciones eliminamos esta restricción, es decir, permitimos que membranas diferentes se correspondiesen con la misma etiqueta. Formalmente diríamos que esta correspondencia entre membranas y etiquetas es sobreyectiva.

La justificación de esta modificación del modelo surge de una necesidad puramente práctica, es decir, en ocasiones interesaría que ciertas membranas diferentes pero con identificador idéntico (y más concretamente sus contenidos) evolucionen bajo la aplica-

ción de un mismo conjunto de reglas. Por tanto, resulta especialmente práctico poder escribir ese conjunto de reglas una única vez y asignarlo a todas las membranas que compartan la misma etiqueta. Si bien esta funcionalidad se implementó en el simulador, finalmente no se hizo uso de la misma pues en ninguno de los escenarios que estudiaremos en el capítulo 5 existen membranas con identificadores compartidos.

Otro cambió que se realizó al modelo tiene relación con la salida que genera el sistema. Como vimos, según el modelo visto en el capítulo 2 marcábamos cierta membrana i_0 que se convertía en el *output* del sistema. En nuestro simulador, sin embargo, distinguiremos tres tipos diferentes de salida que exponemos a continuación.

1. Método 1: Un primer método de salida consistirá en hacer un conteo de todos los objetos en todas las membranas. El método hace además un conteo de los elementos que "salen" del sistema, es decir, atraviesan la piel del sistema y se liberan al entorno. Este método resultará interesante únicamente en aquellos sistemas que cuenten con un número reducido de membranas y de objetos.

2. Método 2: El segundo método de salida consistirá en hacer un conteo en algunas membranas previamente designadas. Ese conteo se hará además para ciertos objetos. Para entender mejor esta forma de generar la salida, vamos a trabajar sobre un ejemplo. Supongamos que queremos saber la cantidad de objetos a contenidos en las membranas con identificador $membrana - ejemplo$. En este caso sumaríamos la multiplicidad del objeto a situado, no sólo directamente en el interior de las membranas con identificador $membrana - ejemplo$, sino también, en el interior de cualquier membrana situada dentro de $membrana - ejemplo$. En el siguiente apartado 3.2 de este mismo capítulo veremos con más detalle cómo indicarle al sistema que debe realizar este tipo de conteo.

Estos fueron los cambios que se introdujeron en el modelo con el objetivo de adaptarlo a las simulaciones y experimentos a desarrollar. A continuación, procedemos a explicar los pasos previos necesarios para lanzar el simulador y codificar el sistema P a ejecutar.

3.2 Pasos previos. Configuración inicial y reglas

Procedemos en este apartado a explicar los pasos previos necesarios antes de proceder a la ejecución del sistema. Parece evidente que, en cualquier tipo de simulación que lancemos, será necesario conocer previamente el alfabeto de objetos V, la estructura de membranas μ, los multiconjuntos correspondientes a cada mebrana $w_i, 1 \leq i \leq n$ (asumiendo que $deg(\mu) = n$), las reglas de evolución y sus prioridades $(R_i, \rho_i), 1 \leq i \leq n$ así como la forma de codificar la salida del sistema de acuerdo a uno de los dos métodos expuestos en el apartado 3.1 de este mismo capítulo.

Para obtener todos los elementos necesarios, se utilizaron dos ficheros *xml* previamente diseñados. El primero consistía en el fichero de configuración del sistema P, es decir, permitía identificar la estructura de membranas (junto con el identificador que correspondía a cada membrana) así como los multiconjuntos iniciales de cada una de ellas. Por otra parte, el segundo fichero permitía identificar el resto de elementos necesarios para construir el sistema. Mencionar además que, para efectuar las lecturas necesarias se utilizó la librería *DOM Parser* para *Java*. El segundo archivo *xml* estaba dividido en las siguientes secciones:

1. Sección *alphabet*: Permitía identificar el alfabeto de objetos V, es decir, los objetos que estarán presentes en las membranas y que evolucionarán bajo la aplicación de las reglas de evolución.

2. Sección *membranes*: Contiene las reglas que corresponden a cada membrana. Cada una de estas reglas vendrá compuesta por su parte izquierda, parte derecha, valor de prioridad y valor de probabilidad o propensión. Por una parte, la prioridad será un valor real, que, a valores más altos identificará una prioridad más alta de su correspondiente regla. Estos valores de prioridad permitirán establecer una ordenación de las reglas, generando de esta forma la relación de orden parcial ρ correspondiente a la membrana. En cuanto a los valores de la probabilidad o propensión se utilizarán al estudiar sistemas estocásticos y su explicación se reserva al capítulo 4 de esta memoria.

3. Sección *output*: Esta sección codifica la salida de acuerdo al segundo método que vimos en el apartado 3.1 de este capítulo.

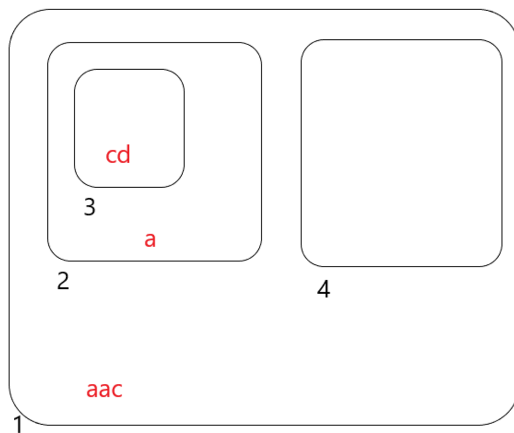
Al terminar de leer estos dos ficheros, tendremos preparado el simulador con su configuración inicial y listo para lanzar la ejecución de las reglas. Mencionar además que el simulador permite desde su inicio leer los dos ficheros mencionados a partir de su ruta, fijar el método de salida del sistema y establecer los pasos de computación a realizar así como el número de ejecuciones.

En el capítulo 2 introdujimos el concepto de paso de computación y cómo suponía partir de una configuración del sistema y pasar a otra configuración potencialmente diferente a la anterior. El hecho de poder fijar el número de pasos desde el principio nos permite estudiar la evolución del sistema hasta cierto momento, lo que resulta atractivo desde el punto de vista de la experimentación. Algo a tener en cuenta es que, es posible que el sistema "termine" antes de alcanzar los pasos fijados por el experimentador. Consideraremos que el sistema ha terminado cuando no se puede aplicar ninguna regla en ninguna de sus membranas.

Por otra parte, el hecho de fijar el número de ejecuciones permite repetir los experimentos y obtener valores medios de todas las ejecuciones. Esto resulta especialmente interesante para estudiar la variabilidad que presenta la ejecución de un sistema en función del algoritmo de selección de reglas que estemos utilizando.

Para terminar, el fichero de configuración del sistema contiene una única sección *model* con las membranas del sistema así como sus contenidos iniciales. Cada membrana se identifica mediante un nodo *membrane* y un atributo *id* de tipo *String* que sirve como identificador o etiqueta de la membrana y que, como vimos en el capítulo 2, puede no ser único. Por otra parte, los nodos hijo pueden ser objetos o membranas. Los objetos se identifican mediante un nodo *BO* y con dos atributos, un atributo *v* de tipo *string* y otro *m* de tipo entero que indica la multiplicidad de ese objeto en la membrana.

Para entender con mayor claridad el propósito de este fichero *xml*, vamos a ver un ejemplo de una representación mediante Diagramas de Venn de un sistema P muy sencillo y su fichero *xml* equivalente. Como podemos ver, el fichero *xml* correspondiente al sistema de la Figura 3.1 permite obtener la estructura de membranas μ del sistema así cómo los multiconjuntos iniciales del mismo. Mencionar el atributo de tipo entero m que aparece en las membranas indica cuantas membranas idénticas tenemos, es decir si por ejemplo, en la membrana con $id = "4"$ tuviésemos $m = "2"$ eso indicaría que tendríamos dos membranas iguales con los mismos contenidos y contenidas en la misma región. Sin embargo, por las necesidades de las simulaciones este valor de m siempre se ha mantenido a 1 para todos los experimentos conducidos.



(a) Sistema P de ejemplo 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<model>
  <config>
    <membrane id="1" m="1">
       <BO v="a" m="2"/>
       <BO v="c" m="1"/>
      <membrane id="2" m="1">
        <BO v="a" m="1"/>
        <membrane id="3" m="1">
          <BO v="c" m="1"/>
          <BO v="d" m="1"/>
        </membrane>
      </membrane>
      <membrane id="4" m="1">
      </membrane>
    </membrane>
  </config>
</model>
```

(b) Código xml equivalente al Sistema P de ejemplo 3.

Figura 3.1: Transformación de un sistema P a fichero xml de configuración

3.3 Selección de reglas y ejecución

Una vez hemos efectuado la lectura de los dos ficheros *xml* descritos en la sección 3.2 podemos pasar a ejecutar las reglas del sistema. Para ello, se hacía un recorrido de la estructura de membranas y se iban ejecutando las reglas de cada una. Por tanto, para cada membrana del sistema se seguía el siguiente esquema de pasos:

1. Del conjunto de reglas correspondientes a la membrana se descartan aquellas que no son aplicables. Consideramos que una regla no es aplicable cuando su parte izquierda no está completamente presente en la membrana (entiéndase completamente presente como que todos los objetos que utiliza están presentes) o cuándo trata de mover algún objeto a una membrana que no está en su interior (posiblemente por haber sufrido una disolución en un paso de computación previo).

2. De ese conjunto de reglas aplicables, las agrupamos por bloques de prioridad. En otras palabras, generamos tantos bloques de prioridad cómo valores diferentes de prioridad existan en el conjunto de reglas aplicables.

3. Para cada bloque de prioridad, empezando siempre por el que contenga las reglas con el nivel más alto de prioridad, procedemos a realizar la selección de las reglas que terminarán por ejecutarse. La selección de estas reglas, o en su defecto el número de veces que se aplicará cada una, vendrá determinado por el algoritmo de aplicación que se empleé en cada ejecución. La descripción de estos algoritmos se reserva al capítulo 4 de esta memoria.

4. Pasamos a la siguiente membrana y repetimos estos pasos hasta que no queden más membranas por examinar en el sistema.

Cuando este proceso se haya completado y hayamos examinado y ejecutado las reglas (las reglas que se puedan ejecutar) en todas las membranas del sistema, entonces, diremos que hemos completado un paso de computación. Cuando termine un paso de computación, pasaremos al siguiente y trataremos de ejecutar tantos pasos como se hayan indicado durante el arranque del simulador (como se explicó en la sección 3.2 de

este capítulo). Sin embargo, como vimos es posible que en algún momento durante la ejecución del sistema este se quede sin reglas que pueda ejecutar. En ese momento consideraremos que el sistema "para" pues sus contenidos no evolucionarán si tratamos de ejecutar más pasos de computación.

3.4 Salida del sistema y recopilación de información

Una vez concluidas las ejecuciones fijadas durante el arranque del simulador, se procedía a generar el *output* del sistema. Como hemos visto en la sección 3.1 de este capítulo, se propusieron dos métodos de salida que suponían introducir una variación en la salida de los modelos estudiados en el capítulo 2. A estos dos métodos cabe añadir un tercer método que al concluir la ejecución, generaba un fichero *xml* de configuración con la misma estructura que el mencionado en la sección 3.2 (es decir, un fichero con la disposición de las membranas y los multiconjuntos que contienen). Esto se hizo para poder lanzar una ejecución con cierto número de pasos y dar la posibilidad de partir de esa configuración (la generada al ejecutar dicho número de pasos) para lanzar otra ejecución. Además esto permitía modificar el conjunto de reglas del escenario y proseguir con una ejecución en la que el escenario evolucionaría de una forma potencialmente diferente a cómo lo hacía antes de modificarlas. Este método de salida, sin embargo, no llegó a utilizarse en los experimentos que expondremos en el capítulo 5.

Los otros dos métodos de salida generaban ficheros *csv* con los resultados. Se eligió este formato por ser el más conveniente a la hora de examinar los resultados y poder realizar un análisis en profundidad de los mismos. Además, se utilizó la librería *OpenCSV* para *Java* con el objetivo de escribir y almacenar los resultados en este formato. Mencionar que, para cualquiera de los dos métodos de salida, al finalizar una ejecución se generaba un fichero con los conteos en cada paso de computación. Es decir, para una ejecución con n pasos de computación generaríamos:

- En el método 1, un fichero con n filas y tantas columnas como objetos distintos en cada membrana en el que mostramos la cantidad presente de cada uno en cada paso de computación.
- En el método 2, un fichero con n filas y tantas columnas como objetos y membranas previamente designadas en el que mostramos la cantidad presente de cada uno en cada paso de computación.

Para concluir, mencionar que estos cambios en la salida y la capacidad de poder recopilar estadísticas para cada paso de computación se implementaron con el objetivo de poder hacer un estudio con mayor detalle de la ejecución completa de los escenarios propuestos.

CAPÍTULO 4

Sistemas estocásticos basados en propensión. Algoritmos de ejecución de las reglas

Procedemos a hacer una introducción de los denominados sistemas P estocásticos. Como ya vimos en el capítulo 1, la computación con membranas trata de imitar a la naturaleza computacional de las operaciones o procesos que tienen lugar en una célula. Si bien una buena parte de la investigación académica actual se ha dedicado a explorar la complejidad de estos sistemas y su poder computacional, los sistemas P también se han utilizado de forma exitosa para modelizar ciertos fenómenos biológicos [6]. Un ejemplo a destacar podría ser el uso de sistemas P estocásticos para simular sistemas de regulación génica que consisten en interacciones proteína-proteína y proteína-ADN que tienen lugar en diferentes niveles de la estructura jerárquica de una célula [18].

Otro ejemplo que resulta de interés es cierto estudio que trató de modelizar metapoblaciones utilizando sistemas P estocásticos [1]. Una metapoblación es un modelo que describe las interacciones y el comportamiento de poblaciones que viven en hábitats fragmentados. Desde un punto de vista ecológico, surge interés en el estudio de la persistencia o la extinción de estas poblaciones, y este estudió trató de plantear una forma de analizarlas mediante un marco de modelización discreta y estocástica.

El éxito que tuvieron estos dos proyectos junto con otros que no nos pararemos a analizar [19, 5], hacen de los sistemas P estocásticos un excelente modelo con el que conducir simulaciones tanto de sistemas biológicos como de ecosistemas complejos con poblaciones elevadas y numerosos factores que pueden afectar a la evolución de esas poblaciones.

Resulta importante recalcar el motivo por el que se implementaron estos dos algoritmos y se probaron con diversos escenarios. Por una parte, como vimos en el capítulo 1, existe un interés teórico en la implementación de algoritmos que emplean constantes estocásticas para seleccionar las reglas a aplicar en cada paso de computación así como en realizar una comparación de los mismos. Por otra parte, existe un objetivo fundamentalmente práctico que consiste en verificar si efectivamente estos algoritmos resultan aptos para escenarios de interés biológico como la expansión del virus COVID-19. Pasamos a continuación a definir formalmente este modelo que ya hemos introducido.

4.1 Sistemas P estocásticos

Al modelo de sistemas P de transición que introdujimos en el capítulo 2 vamos a añadirle un mayor grado de complejidad mediante el uso de valores de las constantes

estocásticas en las reglas de evolución. Los procesos en entornos biológicos que tienen lugar a nivel celular y a nivel molecular suelen ser procesos inherentemente aleatorios [6]. Con el objetivo de trasladar esta aleatoriedad a un sistema P de transición, surgen estos sistemas P estocásticos que pasamos a definir. Un sistema P estocástico es una tupla [17]

$$\Pi = (O, L, \mu, \mathcal{M}_1, ..., \mathcal{M}_n, R_1, ..., R_n)$$

dónde:

- O es un conjunto finito de objetos que especifica las entidades involucradas en el sistema.
- $L = \{l_1, ..., l_n\}$ es un conjunto finito de etiquetas asociadas a las membranas del sistema.
- μ es una estructura de membranas compuesta por $n \geq 1$ membranas que definen las regiones del sistema. La membrana más externa es, de nuevo, la piel del sistema.
- $\mathcal{M}_i = (l_i, w_i, s_i)$ para cada $1 \leq i \leq n$, es la configuración inicial de la membrana i dónde $l_i \in L$ es la etiqueta de la membrana, $w_i \in O^*$ es un multiconjunto finito (cadena) de objetos y s_i es un conjunto finito de cadenas sobre O (nosotros no utilizaremos este conjunto de cadenas en nuestros escenarios).
- $R_{l_k} = \{r_1^{l_k}, ..., r_{m_{l_k}}^{l_k}\}$ para cada $1 \leq k \leq n$ es un conjunto de reglas de evolución. Cada conjunto de reglas R_{l_k} está ligado con las membranas cuya etiqueta sea l_k. Además, estas reglas adoptan la siguiente forma

$$r_i^{l_k} : o_1[o_2]_l \xrightarrow{c_i^{l_k}} o_1^{'}[o_2^{'}]_l$$

dónde o_1, o_2 y $o_1^{'}$, $o_2^{'}$ son multiconjuntos sobre O que representan los objetos que consumen y se producen mediante la aplicación de cada regla. En esencia, esta regla toma la totalidad de los objetos o_1 (en el exterior de la membrana l) y o_2 (contenidos en la membrana l) y los transforma en $o_1^{'}$ y $o_2^{'}$. Otro elemento a destacar es la constante estocástica $c_i^{l_k}$ que se utiliza para calcular la propensión de una regla multiplicándola por el número de "reactivos" presentes en la membrana, es decir, por la multiplicidad de los objetos presentes en la parte izquierda de la regla. Para concluir con esta definición de los sistemas P estocásticos, vamos a comentar una de las modificaciones que introdujimos en el modelo para establecer un orden de prioridad entre las reglas. Para asignar un "nivel" de prioridad a cada regla, se añadió a cada regla un valor real $p_i^{l_k} \in \mathbb{R}^{>0}$ de la siguiente manera

$$r_i^{l_k} : o_1[o_2]_l \xrightarrow{p_i^{l_k}, c_i^{l_k}} o_1^{'}[o_2^{'}]_l$$

Este valor real es una medida de cuán prioritaria es cierta regla de aplicación. Es decir, si tomamos dos reglas $r_i^{l_k}$ y $r_{i'}^{l_k}$ y se verifica que $p_i^{l_k} > p_{i'}^{l_k}$ entonces, diremos que la regla $r_i^{l_k}$ es más prioritaria que $r_{i'}^{l_k}$. Antes de proceder a explicar los dos algoritmos implementados, es importante mencionar otro aspecto a tener en cuenta. Estos algoritmos rompen con el principio de maximalidad que explicamos en el capítulo 2 cuando describimos los sistemas P de transición. Este principio establecía que si en el sistema quedan objetos que pueden evolucionar bajo la aplicación de alguna regla, entonces dichos objetos deben

evolucionar. En el caso de los sistemas P estocásticos, consideramos que este principio no aplica de la misma forma que en los sistemas P de transición y por tanto los algoritmos lo romperán a la hora de seleccionar las reglas a aplicar y el número de veces a aplicarlas. Pasamos a continuación a describir el esquema general de selección y aplicación de reglas así como los dos algoritmos implementados.

4.2 Esquema de selección de reglas. Bloques de prioridad

Algorithm 4.1 Esquema general de ejecución

1: **for all** $l_k \in L$ **do** {Recorremos todas las membranas del sistema P estocástico}
2: $block[] \leftarrow BlockPriority(R_{l_k})$
3: **for** $i = 1$ **to** $i = len(block)$ **do** {Recorremos el *array* de bloques de reglas por prioridad}
4: $b^i_{l_k} \leftarrow block[i]$
5: $Algorithm_j(b^i_{l_k})$ {j podrá valer 1 o 2 en función del algoritmo que se esté utilizando}
6: **end for**
7: **end for**

En la sección 3.3 propusimos un esquema de selección de las reglas a ejecutar en cada paso de computación. Este esquema se basaba en la idea de agrupar las reglas por bloques de prioridad y comenzar a seleccionar y ejecutar siempre tomando las reglas más prioritarias. En el Algoritmo 4.1 proponemos un esquema para ilustrar cómo funciona este proceso de selección de reglas que ayudará a entender mejor el funcionamiento de los dos algoritmos mencionados previamente.

Vemos cómo este procedimiento se sirve de la función $BlockPriority(R_{l_k})$ (Algoritmo 4.2). Esta función toma como entrada el conjunto de reglas R_{l_k} y devuelve un *array* de conjuntos de reglas ordenado por nivel de prioridad. Es decir, la primera posición del *array* contendrá las reglas con el mayor nivel de prioridad, la segunda el conjunto con el segundo mayor nivel, etc. Además, únicamente si una regla es aplicable la incluiremos en un bloque de reglas. Vemos por tanto que el esquema general de ejecución consiste en comenzar haciendo un recorrido de todas las membranas del sistema P estocástico. Para cada membrana l_k tomamos su correspondiente conjunto de reglas R_{l_k}. Para cada conjunto de reglas R_{l_k} utilizamos la función $BlockPriority(R_{l_k})$ que toma el conjunto de reglas como entrada y devuelve como salida un *array* de bloques de reglas por prioridad. Recorremos este *array* y para cada bloque aplicamos el algoritmo que se desee (que como veremos, podrá ser el Algoritmo 1 o el Algoritmo 2). Los algoritmos se encargarán de tomar estos bloques de reglas, calcular las ejecuciones de las reglas que contienen y finalmente ejecutarlas tantas veces como les corresponda a cada una.

Algorithm 4.2 Algoritmo de obtención de bloques de reglas por prioridad

1: **function** $BlockPriority(R_{l_k})$
2: $blocks[]$
3: $ind \leftarrow 1$
4: $blocks[ind] \leftarrow \{\}$ {*blocks* se establece como un *array* de conjuntos}
5: **for all** $r_i^{l_k} \in R_{l_k}$ **do** {Recorremos las reglas del bloque de reglas R_{l_k}}
6:   **if** $applicable(R_{l_k})$ **then** {Únicamente si la regla es aplicable la incluiremos en un bloque}
7:     **for** $j = 1$ **to** $j = ind$ **do** {Recorremos el *array* de conjuntos de reglas}
8:       **if** $blocks[j] \neq \{\}$ **then** {Si el conjunto de reglas actual no está vacío}
9:         $r_{i'}^{l_k} \leftarrow first(blocks[j])$ {Tomamos la primera regla del conjunto mediante $first$}
10:         **if** $p_i^{l_k} = p_{i'}^{l_k}$ **then** {Si la regla actual es igual de prioritaria que la obtenida, la introducimos en el conjunto actual}
11:           $append(blocks[j], r_i^{l_k})$
12:         **else if** $p_i^{l_k} > p_{i'}^{l_k}$ **then** {Si la regla acutal es más prioritaria que la obtenida creamos un nuevo conjunto que insertamos en j y el resto de conjuntos se desplazan una posición a la derecha. Esto se hace mediante la función *insert*}
13:           $insert(blocks[j], \{r_i^{l_k}\})$
14:           $ind \leftarrow ind + 1$
15:         **end if**
16:       **else**
17:         $blocks[j] \leftarrow \{r_i^{l_k}\}$
18:         $ind \leftarrow ind + 1$
19:       **end if**
20:     **end for**
21:   **end if**
22: **end for**
23: **return** *blocks*
24: **end function**

4.3 Algoritmo 1 de aplicación

El primer algoritmo de selección de reglas que se desarrolló fue el que se expone a continuación. Este algoritmo se basa en los valores de propensión para realizar un reparto de los objetos entre las reglas de evolución que son aplicables en cierto paso de computación. Para calcular estos valores de propensión, utilizaremos las constantes estocásticas introducidas en la sección 4.1. Antes de proceder sin embargo, resulta de vital importancia definir el concepto de bloques de reglas con objetos en común en la parte izquierda que vamos a utilizar tanto para este algoritmo como para el algoritmo 2 de aplicación. A cada uno de los bloques de reglas por prioridad $b_{l_k}^n$ obtenidos en el esquema general del Algoritmo 4.1 le corresponde un conjunto de bloques de reglas con objetos en común $BR_{l_k}^n$. Este conjunto estará conformado a por un número determinado de bloques de reglas que identificaremos con un subíndice y un superíndice $BR_{l_k}^{n,j}$ (dónde el subíndice l_k indicará la membrana a la que pertenece el bloque y el superíndice n, j servirá para diferenciar los bloques que se generen por cada bloque por prioridad).

Algorithm 4.3 Algoritmo de obtención de bloques de reglas

1: **function** RuleBlocks($b^n_{l_k}$)
2: $BR^n_{l_k} \leftarrow \{\}$
3: $j \leftarrow 1$
4: **for all** $r^{l_k}_m \in b^n_{l_k}$ **do** {Recorremos todas las reglas de $b^n_{l_k}$}
5: **if** $BR^n_{l_k} = \{\}$ **then** {Si el conjunto $BR^n_{l_k}$ está vacío, creamos un nuevo bloque que contiene la regla actual $r^{l_k}_m$ y lo introducimos en $BR^n_{l_k}$}
6: $BR^{n,j}_{l_k} \leftarrow \{r^{l_k}_m\}$
7: $BR^n_{l_k} \leftarrow BR^n_{l_k} \cup \{BR^{n,j}_{l_k}\}$
8: **else**
9: $new_block = True$
10: **for** $i = 1$ **to** $i = j$ **do** {Recorremos los bloques ya creados}
11: **if** $\exists\, r^{l_k}_{m'} \in BR^{n,i}_{l_k}$ tal que $LHS(r^{l_k}_{m'}) \cap LHS(r^{l_k}_m) \neq \emptyset$ **then** {Si en el bloque actual existe alguna regla que comparte elementos en su parte izquierda con la regla actual, introducimos $r^{l_k}_m$ en dicho bloque}
12: $BR^{n,i}_{l_k} \leftarrow BR^{n,i}_{l_k} \cup \{r^{l_k}_m\}$
13: $new_block = False$
14: **break for**
15: **end if**
16: **end for**
17: **if** new_block **then** {Si ningún bloque tiene alguna regla que comparta elementos con $r^{l_k}_m$ creamos uno nuevo con esta regla y lo introducimos en $BR^n_{l_k}$}
18: $j \leftarrow j + 1$
19: $BR^{j,n}_{l_k} \leftarrow \{r^{l_k}_m\}$
20: $BR^n_{l_k} \leftarrow BR^n_{l_k} \cup \{BR^{n,j}_{l_k}\}$
21: **end if**
22: **end if**
23: **end for**
24: {En función del orden en el que se recorran las reglas, es posible que se generen más bloques de los que se deberían existir. Para evitarlo, hacemos un recorrido adicional de dichos bloques y los comparamos dos a dos. Si existen elementos en común, fusionamos los bloques}
25: **for** $i = 1$ **to** $i = card(BR^n_{l_k}) - 1$ **do**
26: **for all** $r^{l_k}_m \in BR^{n,i}_{l_k}$ **do**
27: **for** $j = i + 1$ **to** $j = card(BR^n_{l_k})$ **do**
28: **for all** $r^{l_k}_{m'} \in BR^{n,j}_{l_k}$ **do**
29: **if** $LHS(r^{l_k}_{m'}) \cap LHS(r^{l_k}_m) \neq \emptyset$ **then**
30: {Fusionamos los bloques y eliminamos $BR^{n,j}_{l_k}$ mediante *remove*}
31: $BR^{n,i}_{l_k} \leftarrow BR^{n,i}_{l_k} \cup BR^{n,j}_{l_k}$
32: $BR^n_{l_k} \leftarrow remove(BR^n_{l_k}, BR^{n,j}_{l_k})$
33: **break for**
34: **end if**
35: **end for**
36: **end for**
37: **end for**
38: **end for**
39: **return** $BR^n_{l_k}$
40: **end function**

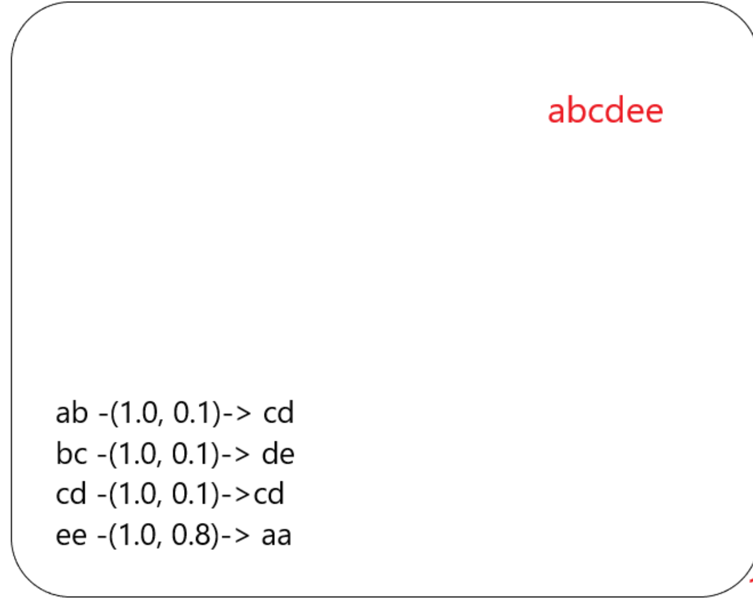


Figura 4.1: Sistema P de ejemplo para formación de bloques.

Este algoritmo recibe cómo entrada el uno de los conjuntos de bloques de reglas por prioridad $b^n_{l_k}$ asociado a cierta membrana con etiqueta l_k y devuelve como salida un conjunto que contiene los diferentes bloques de reglas con objetos en común generados. En términos informales, un bloque de reglas con objetos en común es un conjunto de reglas entre las que existe cierto "solape" , es decir, contendrá aquellas reglas que comparten algún objeto en su parte izquierda pero sólo exigiremos que este solape se de entre una pareja de reglas dentro del conjunto. El algoritmo de obtención de estos bloques de reglas es el Algoritmo 4.3.

Para aportar claridad a este proceso de obtención de los bloques de reglas, vamos a proponer un ejemplo de cómo se generarían a partir del sistema P estocástico que aparece en la Figura 4.1. En este sistema podemos ver que tenemos una única membrana con etiqueta 1 y un multiconjunto inicial *abcdee*. En primer lugar, obtendríamos los bloques de reglas por prioridad. Como todas las reglas son igualmente prioritarias (y todas son aplicables), entonces, únicamente tendremos un bloque con la forma $b^1_1 = \{ab \xrightarrow{1.0,0.1} cd, bc \xrightarrow{1.0,0.1} de, cd \xrightarrow{1.0,0.1} cd, ee \xrightarrow{0.8} aa\}$. Con este bloque llamaríamos a alguno de los algoritmos y se procedería a calcular los bloques de reglas con objetos en común en su parte izquierda. El proceso a seguir sería el siguiente.

1. Para la regla $ab \xrightarrow{1.0,0.1} cd$, ningún conjunto se ha formado todavía por lo que creamos $BR^{1,1}_1 = \{ab \xrightarrow{1.0,0.1} cd\}$ y lo introducimos en BR^1_1.

2. Para la regla $bc \xrightarrow{1.0,0.1} de$, vemos que tiene en común la b con la regla ya incluida en $BR^{1,1}_1$ por tanto $BR^{1,1}_1 = \{ab \xrightarrow{1.0,0.1} cd, bc \xrightarrow{1.0,0.1} de\}$.

3. Para la regla $cd \xrightarrow{1.0,0.1} cd$, vemos que tiene en común la c con la última regla introducida en $BR^{1,1}_1$ luego $BR^{1,1}_1 = \{ab \xrightarrow{1.0,0.1} cd, bc \xrightarrow{1.0,0.1} de, cd \xrightarrow{1.0,0.1} cd\}$

4. Para la regla $ee \xrightarrow{1.0,0.8} aa$, vemos que no tiene en común ningun elemento con las partes izquierdas de las reglas de $BR_1^{1,1}$. Por tanto, generamos $BR_1^{1,2} = \{ee \xrightarrow{1.0,0.8} aa\}$ y lo introducimos en BR_1^1.

Al terminar este proceso, tendríamos que $BR_1^1 = \{BR_1^{1,1}, BR_1^{1,2}\}$, es decir, a esta membrana le corresponden dos bloques de reglas. Finalmente, haríamos un recorrido adicional del conjunto para asegurarnos de que los bloques se han generado correctamente. En este caso, el conjunto BR_1^1 quedaría inalterado. Una vez definido este concepto de bloque de regla, el algoritmo 1 de aplicación es el que aparece en el Algoritmo 4.4 a continuación.

Algorithm 4.4 Algoritmo 1 de aplicación

1: **procedure** Algorithm1($b_{l_i}^n$)
2: $BR_{l_i}^n = RuleBlocks(b_{l_i}^n)$ {Obtenemos los bloques de reglas de acuerdo a los objetos que comparten en su parte izquierda}
3: **for all** $BR_{l_i}^{n,j} \in BR_{l_i}^n$ **do** {Recorremos los bloques de reglas obtenidos}
4: **for all** $r_m^{l_i} \in BR_{l_i}^{n,j}$ **do** {Recorremos las reglas de cada bloque}
5: $exec[]$
6: $ind \leftarrow 1$
7: **for all** $e_s \in LHS(r_m^{l_i})$ **do** {Recorremos los objetos en la parte izquierda de la regla actual}
8: $mult_mem \leftarrow M_{l_i}(e_s)$ {Multiplicidad de e_s en la membrana actual l_i}
9: $mult_lh \leftarrow M_{LHS(r_m^{l_i})}(e_s)$ {Multiplicidad de e_s en la parte izquierda de $r_m^{l_i}$}
10: $num_rules \leftarrow card(\{r_m^{l_i} \in BR_{l_i}^{n,j} \mid e_s \in LHS(r_m^{l_i})\})$ {Número de reglas en $BR_{l_i}^j$ en las que aparece e_s}
11: $exec[ind] \leftarrow (mult_mem \cdot c_m^{l_k})/(mult_lh \cdot num_rules)$ {Número de ejecuciones de $r_m^{l_i}$ de acuerdo a la multiplicidad de e_s}
12: $ind \leftarrow ind + 1$
13: **end for**
14: $n_exec \leftarrow min(exec)$ {El número de ejecuciones de $r_m^{l_k}$ se obtiene como el mínimo de los números de ejecuciones calculados en base a los objetos presentes en su parte izquierda}
15: $n_exec_def \leftarrow ObtainExecutions(n_exec)$ {El número de ejecuciones n_exec puede ser decimal, obtenemos el número entero mediante el algoritmo 4.5}
16: **for** $l = 1$ **to** $l = n_exec_def$ **do** {Aplicamos la regla tantas veces como indique n_exec_def}
17: $apply_rule(r_m^{l_k})$
18: **end for**
19: **end for**
20: **end for**
21: **end procedure**

Algorithm 4.5 Algoritmo para la obtención de un número entero de ejecuciones

1: **function** $ObtainExecutions(k)$
2: $rand = random(0, 1)$ {Generamos aleatorio entre 0 y 1}
3: $d = decimal(k)$ {Tomamos la parte decimal del argumento k}
4: $k_e = int(k)$ {Tomamos la parte entera del argumento k}
5: **if** $rand \leq d$ **then**
6: **return** $k_e + 1$ {Si $rand$ menor o igual que la parte decimal devolvemos $k_e + 1$ ejecuciones}
7: **else**
8: **return** k_e {Si $rand$ mayor que la parte decimal devolvemos k_e ejecuciones}
9: **end if**
10: **end function**

Para tratar de aportar una mayor claridad a la explicación de este algoritmo, vamos a proponer dos ejemplos muy sencillos. El primero de ellos es el que aparece en la Figura 4.2 (mencionar que los enteros 1,2 y 3 que aparecen delante de las reglas sirve simplemente para identificarlas y facilitar la explicación que sigue) . En este caso vemos como tenemos un multiconjunto inicial a^{30} que puede evolucionar en base a tres reglas diferentes. En primer lugar, comenzaríamos generando un bloque de reglas en base a su prioridad. Como las tres reglas tienen el mismo valor de prioridad, se insertarían en un sólo bloque.

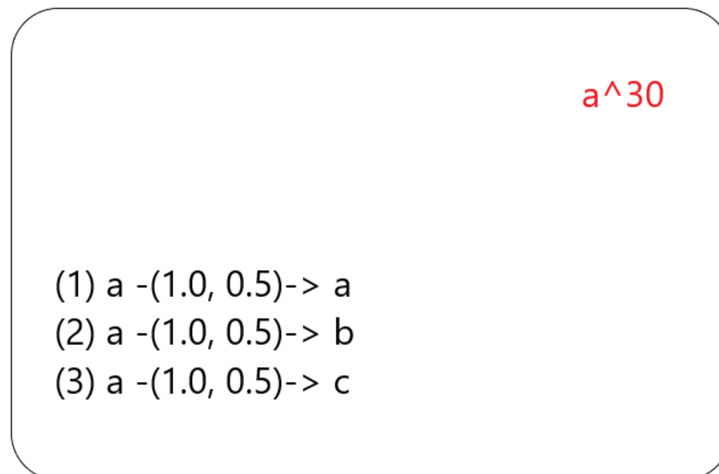


Figura 4.2: Ejemplo 1 para Algoritmo 1.

Seguidamente, tomaríamos este bloque de reglas al mismo nivel de prioridad y generaríamos otros bloques adicionales en base a las reglas que compartan objetos en su parte izquierda. De nuevo, en este caso, cómo únicamente tenemos reglas que actúan sobre a generamos un sólo bloque que contiene a las tres reglas. Teniendo en cuenta que la constante estocástica de cada regla es el número real 0.5 que aparece en la Figura 4.2, calcularíamos las ejecuciones de cada una de la siguiente manera:

- Para la regla r_1^1 obtendríamos $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones.

- Para la regla r_2^1 obtendríamos $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones.
- Para la regla r_3^1 obtendríamos $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones.

Es decir, se aplicaría un total de cinco veces cada regla y se pasaría al siguiente paso de computación. Este ejemplo es una muestra muy sencilla de cómo funciona este algoritmo y para comprenderlo mejor, vamos a proponer otro ejemplo en la Figura 4.3 que supone limitar la aplicación de una regla. En este caso vemos como tenemos un multiconjunto inicial $a^{30}b$ y tres reglas que pueden hacerlo evolucionar. Comenzamos generando los bloques de reglas en base a su prioridad introduciéndolas todas en el mismo bloque (pues todas tienen valor 1.0 de prioridad).

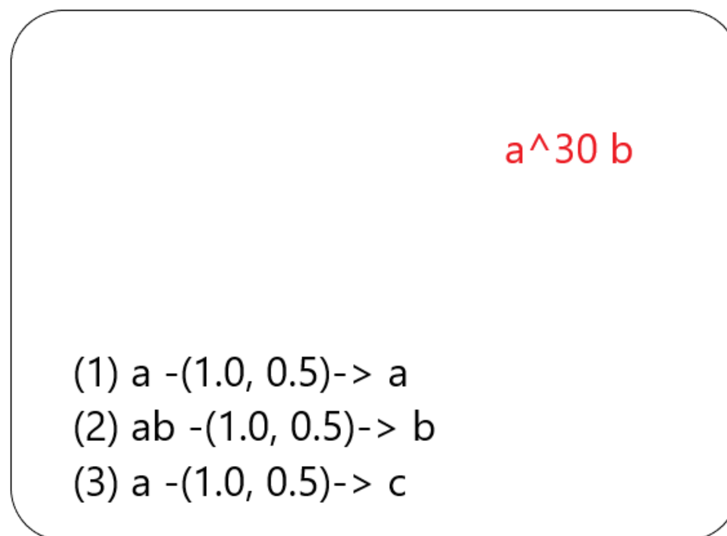


Figura 4.3: Ejemplo 2 para Algoritmo 1.

Tomamos este bloque de reglas más prioritarias y generamos los bloques de acuerdo a si las reglas comparten objetos en su parte izquierda o no. En este caso ocurre lo mismo que en el anterior, es decir, se genera un único bloque que contiene las tres reglas a aplicar. Partiendo de los valores de las constantes estocásticas que aparecen en la Figura 4.3 calculamos las ejecuciones de cada regla de la siguiente manera:

- Para la regla r_1^1 obtendríamos $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones.
- Para la regla r_2^1 obtendríamos por una parte $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones en base a la multiplicidad de a y $\frac{1}{1} \cdot 1 \cdot 0.5 = 0.5$ ejecuciones en base a la multiplicidad de b.
- Para la regla r_3^1 obtendríamos $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones.

Como vemos a partir de estos cálculos, las reglas 1 y 3 seguirían ejecutándose las mismas veces que en el ejemplo anterior, sin embargo, el número de aplicaciones de la regla 2 cambia. El hecho de haber introducido una b en el multiconjunto inicial (y en la parte izquierda de esta regla) nos obliga a tenerla en cuenta a la hora de calcular el número de ejecuciones. Como vimos al principio de esta sección, al calcular el número de ejecuciones de una regla, siempre nos quedábamos con el número mínimo, es decir, para el caso que respecta 0.5 ejecuciones para la regla 2.

Queda ahora comprender el significado de un número no entero de ejecuciones. Cuando obtenemos un número k de ejecuciones posiblemente no entero (cuya parte entera denotaremos como k_e), entonces, el Algoritmo 4.5 procede de la siguiente manera:

1. Tomamos la parte decimal d de k.
2. Generamos un número aleatorio r entre 0 y 1.
3. Si $r \leq d$ a la regla le corresponde un número $k_e + 1$ de ejecuciones.
4. Si $r > d$ a la regla le corresponde un número k_e de ejecuciones.

En el ejemplo propuesto, tendríamos $k = 0.5$, $k_e = 0$ y $d = 0.5$, generaríamos un número aleatorio r entre 0 y 1. Si por ejemplo, $r = 0.02535$ entonces ejecutaríamos la regla un total de $k_e + 1 = 0 + 1 = 1$ vez. Por contra, si $r = 0.7587$ entonces ejecutaríamos la regla 0 veces, es decir, quedaría sin ejecutarse. Mencionar además que, como explicamos al principio de este mismo capítulo, estos algoritmos rompen el principio de maximalidad pues, en ambos casos, al concluir un paso de computación siguen quedando $a's$ que pueden evolucionar bajo la aplicación de alguna regla pero no lo hacen y pasan inalteradas al siguiente paso.

4.4 Algoritmo 2 de aplicación

Algorithm 4.6 Algoritmo 2 de aplicación

1: **procedure** Algorithm2($b_{l_i}^n$)
2: $BR_{l_i}^n = RuleBlocks(b_{l_i}^n)$ {Obtenemos los bloques de acuerdo a los objetos que comparten en su parte izquierda}
3: **for all** $BR_{l_i}^{n,j} \in BR_{l_i}^n$ **do** {Recorremos los bloques de reglas}
4: $subint[0] \leftarrow 0.0$
5: $ind \leftarrow 1$
6: **for all** $r_m^{l_i} \in BR_{l_i}^{n,j}$ **do** {Recorremos las reglas de cada bloque}
7: $subint[ind] \leftarrow subint[ind-1] + \left(c_m^{l_i} / \sum_{\forall r_{m'}^{l_k} \in BR_{l_i}^j} c_{m'}^{l_k} \right)$ {Asignamos a cada regla un subintervalo que corresponde a su probabilidad}
8: $ind \leftarrow ind + 1$
9: **end for**
10: $rand \leftarrow random(0,1)$ {Generamos un aleatorio entre 0 y 1}
11: **for** $l = 1$ **to** $l = ind - 1$ desde el índice 1 **do** {Recorremos $subint$}
12: **if** $rand >= subint[l-1]$ **or** $rand < subint[l]$ **then**
13: {Si el aleatorio está en los valores comprendidos en $[subint[l-1], subint[l])$ entonces la regla $l-ésima$ debe ser ejecutada}
14: $apply_rule(r_l^{l_i})$
15: **break for**
16: **end if**
17: **end for**
18: **end for**

El segundo algoritmo de selección de reglas que se desarrolló fue el que pasamos a exponer. Este algoritmo se basa de nuevo en los valores de las constantes estocásticas de

las reglas para calcular un valor de probabilidad de ejecución para cada una. De forma esquemática, si tuviésemos un sistema P estocástico el algoritmo sería el Algoritmo 4.6.

De nuevo, para aportar claridad a la explicación del algoritmo, vamos a ver un ejemplo que aparece en la Figura 4.4. En este ejemplo partimos de un multiconjunto inicial a^{30} y tenemos un total de tres reglas que pueden hacerlo evolucionar (las reglas aparecen numeradas simplemente para entender más fácilmente la explicación que sigue). Empezamos generando un bloque de reglas de acuerdo a la prioridad de las mismas, en este caso como todas tienen el mismo valor de prioridad se incluyen en el mismo bloque. Seguidamente generamos los bloques de reglas en función de los objetos que comparten en su parte izquierda. De la misma manera que en ejemplos anteriores, las reglas comparten parte izquierda y se incluyen en el mismo bloque.

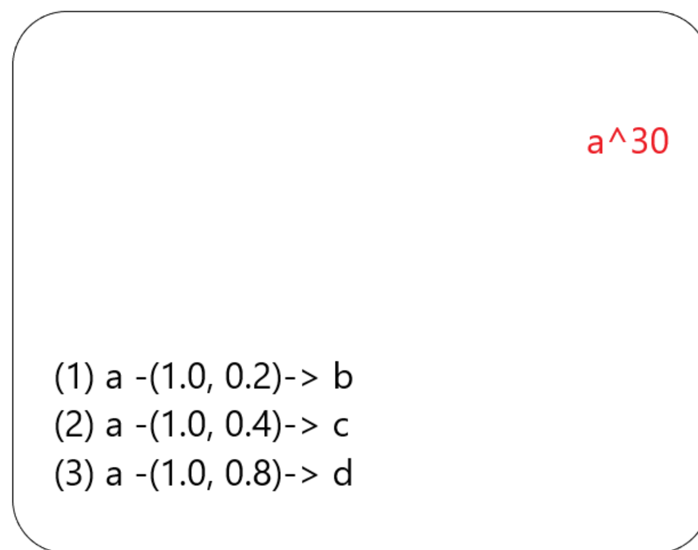


Figura 4.4: Ejemplo 1 para Algoritmo 2.

A partir de los valores de las constantes estocásticas, el *array subint* quedaría de la siguiente manera.

- Para la regla r_1^1, $P(r_1^1) = 0.2/(0.2+0.4+0.8) = 0.142857$, luego $subint[1] = subint[0] + 0.142857 = 0.0 + 0.142857 = 0.142857$.
- Para la regla r_2^1, $P(r_2^1) = 0.4/(0.2+0.4+0.8) = 0.285714$, luego $subint[2] = subint[1] + 0.285714 = 0.428571$.
- Para la regla r_3^1, $P(r_3^1) = 0.8/(0.2+0.4+0.8) = 0.571429$, luego $subint[3] = subint[2] + 0.571429 = 1.0$.

Este *array subint* se podría interpretar también como que a cada regla le corresponden los siguientes subintervalos de probabilidad

- Para la regla r_1^1, $[0.0, 0.142857]$.
- Para la regla r_2^1, $[0.142857, 0.428571]$
- Para la regla r_3^1, $[0.428571, 1.0]$

A continuación se generaría un número aleatorio entre 0 y 1, si el número es 0.1 entonces se ejecutaría la regla 1, si el número fuese 0.3 se ejecutaría la regla 2 y, si fuese 0.8 se ejecutaría la regla 3. De este algoritmo cabe destacar que el hecho de que por cada bloque de reglas se lanza una única ejecución al contrario de lo que ocurría en el algoritmo 1 en el que en promedio se esperaría que se lanzase más de una ejecución por bloque de reglas. Por tanto, a la hora de conducir experimentos con este algoritmo esto se deberá tener en cuenta para lanzar un número mayor de pasos de computación por ejecución. Otro aspecto a considerar es que, de nuevo, el algoritmo rompe el principio de maximalidad pues a pesar de restar $a's$ que pueden evolucionar bajo la aplicación de cierta regla, quedan inalteradas y pasan al siguiente paso de computación.

CAPÍTULO 5
Escenarios. Experimentación y resultados

En este capítulo vamos a exponer todo el proceso de experimentación y análisis de los resultados que se llevó a cabo una vez el simulador estuvo listo para acometer dicha tarea. El capítulo se dividirá en diferentes secciones en las que expondremos los escenarios a tratar, los resultados extraídos de la simulación y un análisis en detalle de aquellos aspectos que consideramos de mayor interés en cada caso. Además, para cada escenario analizaremos los resultados extraídos utilizando los dos algoritmos expuestos en el capítulo 4 junto con una comparación de los mismos. Trataremos además de dar una justificación detallada de todos los resultados que se vayan obteniendo, es decir trataremos de dar sentido y razonar las tendencias que presenten los escenarios con cada algoritmo así como comportamientos anómalos o llamativos que puedan resultar de interés.

Concluiremos el capítulo con una exposición de un escenario que simula la propagación del virus COVID-19 en un entorno que contempla un total de 500 unidades familiares. La simulación y obtención de resultados en este escenario concreto resultaba de especial interés en este trabajo además de constituir uno de los objetivos a cumplir. Por todo ello, dedicaremos el grueso de este capítulo a entenderlo correctamente y hacer una interpretación lo más rica posible de los resultados. Incluimos además en el material suplementario de este TFG los ficheros *xml* diseñados para obtener la configuración inicial de los escenarios que pasamos a analizar.

5.1 Escenario 1

Comenzamos con el Escenario 1 proponiendo la siguiente definición formal del mismo:

$$\Pi = (O, L, \mu, \mathcal{M}_1, R_1)$$

dónde:

- $O = \{a, b, c\}$
- $L = \{1\}$
- $\mu = [\,]_1$
- $\mathcal{M}_1 = (1, a^{30})$

- $R_1 = \{r_1^1 : a \xrightarrow{1.0,0.5} ab, r_2^1 : a \xrightarrow{1.0,0.5} ac, r_3^1 : a \xrightarrow{1.0,0.5} ab\}$

Una representación visual del Escenario 1 mediante diagramas de Venn es la que puede verse en la Figura 5.1. Como se desprende del diagrama y la descripción formal nos encontramos frente a un sistema P con una única membrana y un multiconjunto inicial a^{30} que puede evolucionar bajo la aplicación de tres reglas de evolución. Podemos ver como estas reglas tienen un valor idéntico de la constante estocástica. Mencionar además que en los diagramas de los escenarios que aparecen en este capítulo, omitiremos los valores de las prioridades y mostraremos únicamente los valores de las constantes estocásticas. Los valores de prioridad aparecerán en la definición formal de cada escenario.

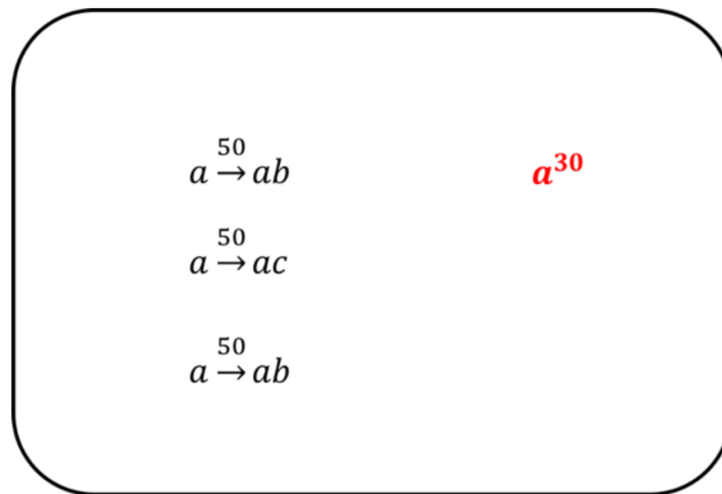


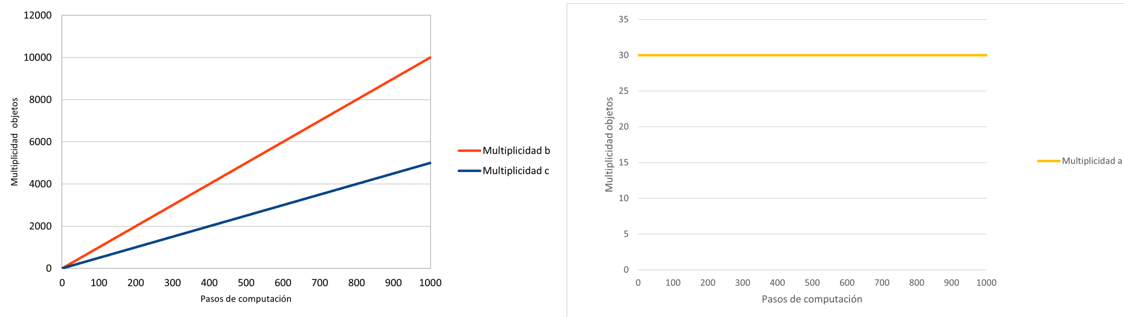
Figura 5.1: Escenario 1.

Pasamos a hacer un análisis de los resultados obtenidos. Antes de proceder, mencionar que el método de salida escogido fue el método 1 que explicamos en la sección 3.4, es decir, se hizo un conteo de cada objeto en la única membrana presente para todos los pasos de computación fijados.

5.1.1. Escenario 1 - Algoritmo 1

El experimento que se lanzó para este primer escenario utilizando el Algoritmo 1 fue un experimento de 500 ejecuciones del sistema con 1000 pasos de computación por ejecución. Una vez concluida la experimentación se generó un fichero csv que calculaba los valores medios de cada objeto en cada paso de computación a partir de los valores generados durante las ejecuciones. La representación de estos valores medios de progresión frente a los pasos de computación pueden verse en la Figura 5.2 a continuación.

De estas progresión resultan de especial interés varios aspectos. Por una parte, vemos que la multiplicidad del objeto a viene representado como una recta paralela al eje de abscisas, es decir, se mantiene constante durante todos los pasos de computación. Más concretamente, la multiplicidad de a $M_1(a)$ permanece en todo momento como $M_1(a) = 30$ (Figura 5.2.b). Esto se puede ver fácilmente analizando con mayor detenimiento las reglas de este escenario. Como se puede ver, toda regla consume una a en su parte izquierda pero a cambio genera una nueva a mediante su parte derecha. En consecuencia, independientemente de qué regla o reglas se ejecuten y cuántas veces lo hagan, siempre tendremos el mismo número de $a's$ en la membrana.



(a) Progresión objetos b y c Escenario 1 - Algoritmo 1 **(b)** Progresión objeto a Escenario 1 - Algoritmo 1

Figura 5.2: Progresión objetos Escenario 1 - Algoritmo 1

Por otra parte, en este sistema tenemos un único bloque de reglas por prioridad ya que, como se desprende de la definición formal ninguna regla es más prioritaria que otra y también existe únicamente un bloque de reglas por los objetos que comparten en su parte izquierda (ya que todas las reglas tienen una a en su parte izquierda). Por todo esto, todas las reglas generarán el mismo número de ejecuciones dado por $\frac{1}{3} \cdot 30 \cdot 0.5 = 5$ ejecuciones. Si cada paso supone ejecutar 5 veces cada regla entonces, es de esperar que generemos 10 $b's$ y 5 $c's$ por paso. Una forma de verificar esto es realizar una ajuste lineal mediante mínimos cuadrados de la multiplicidad de cada objeto frente a los pasos de computación. Dicho ajuste puede verse en la Figura 5.3 a continuación.

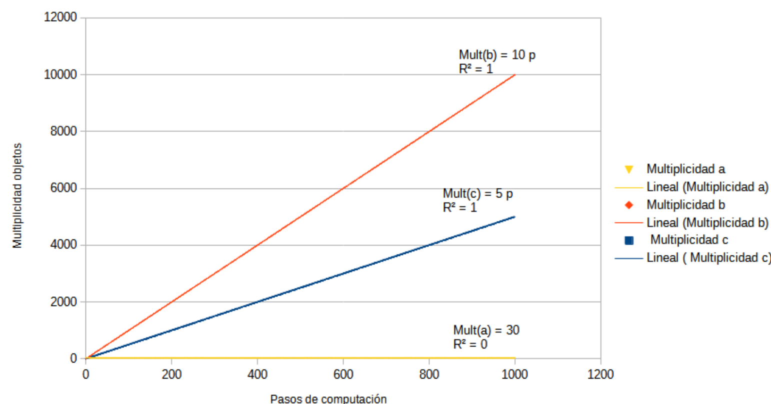


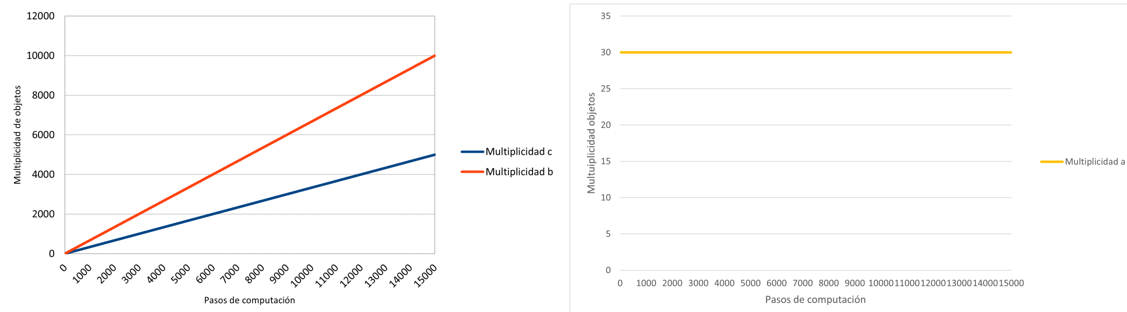
Figura 5.3: Ajuste por mínimos cuadrados Escenario 1 - Algoritmo 1.

De este ajuste vemos con la línea de tendencia para obtener la multiplicidad el objeto b es $M_1(b) = 10p$ dónde p son los pasos de computación. Lo mismo puede verse para obtener la multiplicidad del objeto c que es $M_1(c) = 5p$. Todo esto parece corroborar la hipótesis de que en cada paso se generan 10 $b's$ y 5 $c's$.

5.1.2. Escenario 1 - Algoritmo 2

Antes de lanzar las ejecuciones correspondientes al Escenario 1 utilizando el Algoritmo 2, se planteó la cuestión de cómo elegir el esquema de ejecuciones y pasos por ejecución que permitiese comparar los resultados obtenidos con los del Algoritmo 1. Teniendo en cuenta que, el Algoritmo 2 ejecutaba una única regla por bloque de reglas, parece evidente que para conseguir el mismo número de objetos que en el paso 1000 con el Algoritmo 1, serán necesarios un número mayor de pasos.

En concreto, en el subapartado anterior vimos que, por cada paso de computación generábamos 10 $b's$ y 5 $c's$, en otras palabras, un total de 15 objetos nuevos por paso. Por tanto, para conseguir el mismo número de objetos mediante el segundo algoritmo, serán necesarios 15 veces más pasos que con el primero. Por todo esto, el esquema elegido fueron 500 ejecuciones de 15000 pasos por ejecución. La valores medios de la progresión pueden verse en la Figura 5.4 a continuación.



(a) Progresión objetos b y c Escenario 1 - Algoritmo 2 **(b)** Progresión objeto a Escenario 1 - Algoritmo 2

Figura 5.4: Progresión objetos Escenario 1 - Algoritmo 2

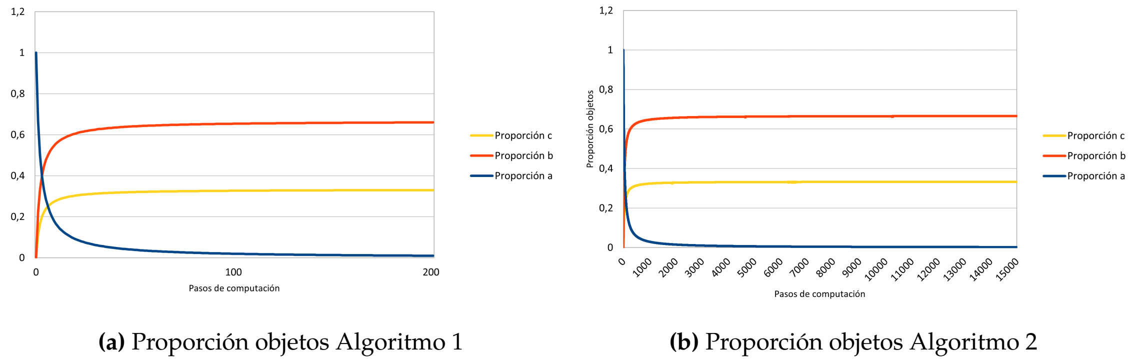
De esta gráfica destacan varios aspectos que vamos a pasar a comentar. Por una parte, seguimos teniendo un número constante de 30 $a's$ durante todos los pasos de computación. Esto es así porque, en cada paso ejecutamos una única regla pero, como vimos, cada regla consume una a y genera una nueva a lo que impide que el número de $a's$ se reduzca con respecto al inicial.

Además, se sigue apreciando que el número de $b's$ generadas es el doble que el número de $c's$. Esto puede explicarse teniendo en cuenta las probabilidades que asignan a cada regla mediante este algoritmo. En este caso, tendríamos tres reglas con constante estocástica de 0.5 cada una, por tanto, el cálculo que propone el algoritmo asignaría una probabilidad del 33.33 % para cada regla. Esto se traduce a que en cada paso de computación tenemos una probabilidad del 66.67 % de generar una b y una probabilidad del 33.33 % de generar una c. En consecuencia, en una ejecución de 15000 pasos, el número de $b's$ debería aproximarse a 10000 y el número de $c's$ a 5000.

5.1.3. Comparación Algoritmo 1 - Algoritmo 2

Para comparar como actúan ambos algoritmos en este escenario, podríamos pararnos a analizar las gráficas de las Figuras 5.2 y 5.4. En este caso, y como ya se ha explicado, estas dos gráficas presentan una tendencia similar en la progresión de los objetos. Otra forma de hacer esta comparación consistiría en calcular la proporción de cada objeto en cada paso de computación para cada algoritmo. Es decir, calcular el número total de objetos en cada paso de computación y dividir la cantidad de cada objeto entre ese total para obtener la proporción de dicho objeto. Si obtenemos dichas proporciones y las graficamos frente a los pasos de computación, se obtienen las gráficas que aparecen en la Figura 5.5 a continuación.

De estas dos gráficas puede verse como la tendencia de los objetos es la misma para ambos algoritmos, con la única diferencia de que el algoritmo 2 necesita un mayor número de pasos para aproximar esta tendencia. El descenso en picado de la proporción de a ocurre porque, en principio, es el único objeto que existe en la membrana (de ahí que su proporción sea 1.0) pero rápidamente el número de $b's$ y $c's$ aumentan hasta superar el número de $a's$. En conclusión, por todo lo visto podemos afirmar que el Algoritmo 1 y el Algoritmo 2 presentan un comportamiento equivalente para este escenario en concreto.



(a) Proporción objetos Algoritmo 1 **(b)** Proporción objetos Algoritmo 2

Figura 5.5: Proporción por objeto para cada algoritmo

5.2 Escenario 2

Procedemos con el Escenario 2 planteando en primer lugar una descripción formal del mismo:

$$\Pi = (O, L, \mu, \mathcal{M}_1, R_1)$$

dónde:

- $O = \{a, b, c\}$
- $L = \{1\}$
- $\mu = [_1]_1$
- $\mathcal{M}_1 = (1, a^{3000})$
- $R_1 = \{r_1^1 : a \xrightarrow{1.0,0.2} aab, r_2^1 : aa \xrightarrow{1.0,0.5} c, r_3^1 : ac \xrightarrow{1.0,0.5} ab\}$

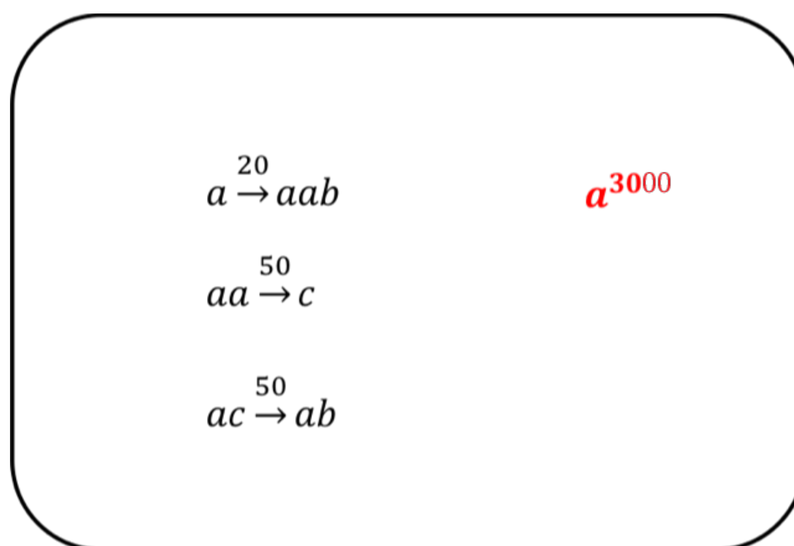


Figura 5.6: Escenario 2.

Una representación visual del Escenario 2 mediante diagramas de Venn es la que puede verse en la Figura 5.6. Como se desprende de la descripción formal y del diagrama,

de nuevo, nos encontramos frente a un sistema P con una única membrana, un multiconjunto inicial a^{3000} y tres reglas que pueden hacerlo evolucionar (como vemos también, al mismo nivel de prioridad). Estas reglas vienen acompañadas de valores de las constantes estocásticas que, en este caso, no son todos iguales lo que en principio generará un reparto menos uniforme que el que vimos en el Escenario 1.

Procedemos a hacer un análisis de los resultados obtenidos empleando los diferentes algoritmos. De nuevo, el método de salida escogido fue el método 1 con el objetivo de obtener las progresiones de los objetos en la membrana y facilitar una comparación entre los dos algoritmos.

5.2.1. Escenario 2 - Algoritmo 1

El experimento que se lanzó con este escenario para el Algoritmo 1 consistió en un experimento de 500 ejecuciones con 1000 pasos de computación para cada una. Sin embargo, algo que resultaba llamativo al examinar los resultados fue que ninguna de las 500 ejecuciones lograba "superar" los 166 pasos de computación, es decir, ninguna ejecución lograba mantener un número de $a's$ superior a 0 durante más de 166 pasos. Esta progresión de los diferentes objetos es la que puede observarse en la Figura 5.7 a continuación. Mencionar que, como ninguna ejecución conservaba a's hasta más allá de 166 pasos, únicamente hemos incluido 200 pasos de computación en el eje de abscisas.

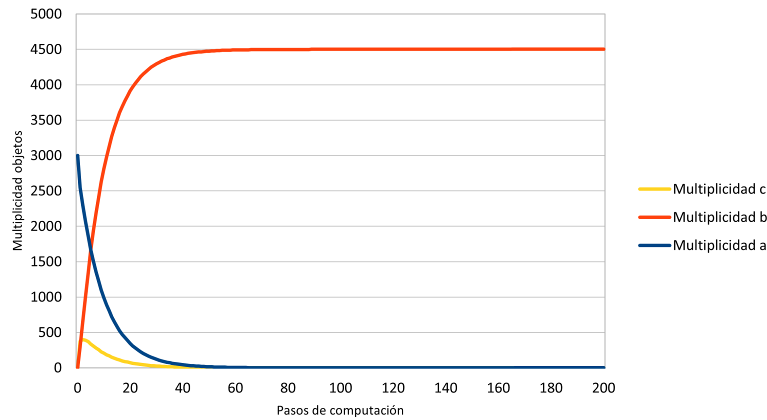


Figura 5.7: Progresión objetos Escenario 2 - Algoritmo 1.

Como puede verse de esta gráfica, la multiplicidad del objeto a comienza a decaer rápidamente y a aproximarse a 0 a partir de los 50 pasos. Vamos a pararnos a razonar el motivo de este comportamiento. El primer paso de computación siempre será igual en cualquier ejecución que lancemos, es decir, ocurrirá lo siguiente:

- Para la regla $r_1^1 : a \xrightarrow{0.2} aab$ tendremos $\frac{1}{2} \cdot 3000 \cdot 0.2 = 300$ ejecuciones.
- Para la regla $r_2^1 : aa \xrightarrow{0.5} c$ tendremos $\frac{1}{2} \cdot \frac{1}{2} \cdot 3000 \cdot 0.5 = 375$ ejecuciones.

Por lo que, en el siguiente paso de computación tendremos $3000 - (375 \cdot 2) + 300 = 2550$ $a's$, 300 $b's$ y 375 $c's$. A partir de este segundo paso todas las reglas del sistema pueden ser ejecutadas puesto que tenemos $a's$ y $c's$ suficientes para ello. De todas las reglas que ahora son aplicables, puede verse que la primera regla r_1^1 es la única que supone aumentar en una unidad el número de $a's$ mientras que la regla r_2^1 supone disminuirlo en dos y la regla r_3^1 dejarlo constante. El hecho de que la segunda regla r_2^1 tenga un valor

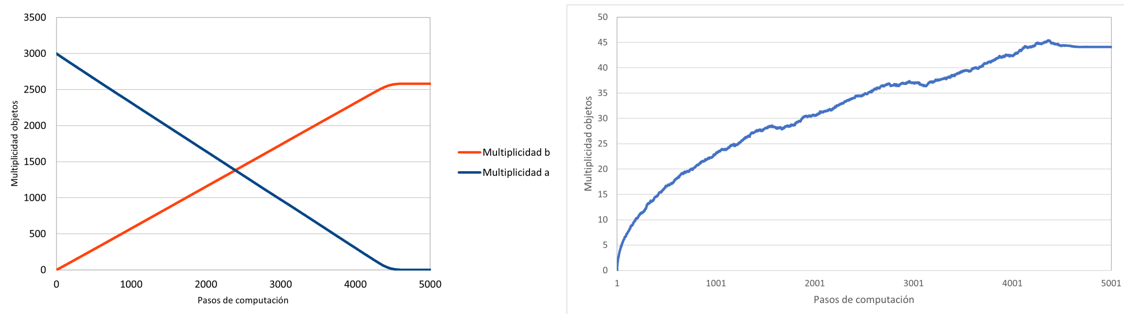
de la constante estocástica mayor que la primera hace que reciba un número mayor de ejecuciones y, por tanto, la tendencia sea a que el número de $a's$ termine por agotarse.

Otro aspecto a destacar es la progresión del objeto b que, como se puede ver, experimenta un crecimiento hasta aproximadamente 4500 $b's$. Este tendencia monótona creciente puede explicarse teniendo en cuenta que tenemos dos reglas que generan una b cada vez que se ejecutan y además, la b no aparece en ninguna parte izquierda por lo que su número nunca se reducirá.

Por último, otra tendencia que se observa es una disminución progresiva del número de $c's$. La causa de este descenso en la multiplicidad es debida a la forma de repartir las ejecuciones entre las dos últimas reglas. Por una parte, la regla r_2^1 supone aumentar en uno el número de $c's$ mientras que la regla r_3^1 supone hacerlo disminuir en uno. Lo que ocurrirá en este escenario es que en un principio, el número de $c's$ aumentará pues r_2^1 se ejecutará un número mayor de veces que r_3^1 (pues r_3^1 vendrá condicionada por la multiplicidad de c que será menor que la de a). Eventualmente, el número de $a's$ se reducirá lo suficiente como para que r_2^1 se ejecute un número menor de veces que r_3^1, por lo que se iniciará esa tendencia al descenso de c.

5.2.2. Escenario 2 - Algoritmo 2

El experimento que se lanzó con el Algoritmo 2 consistió en 500 ejecuciones de 5000 pasos cada una. El objetivo de lanzar el experimento con este esquema de ejecuciones y pasos por ejecución fue el tratar de conseguir un número de objetos en la membrana similar al que se obtuvo con el Algoritmo 1. Los resultados de las progresiones de los objetos son los que pueden verse en las gráficas de la Figura 5.8. De estas progresiones podemos destacar varios aspectos. Por una parte, ninguna ejecución logró superar el paso 4772 con un número de $a's$ superior a 0, lo que explicaría como, por ejemplo, la progresión de las $b's$ se detiene y la línea queda paralela al eje de abscisas. Además, como ya hemos explicado, esta tendencia es monótona creciente puesto que no aparece ninguna b en ninguna parte izquierda de las reglas y por tanto su número nunca se reduce.



(a) Progresión objetos a y b Escenario 2 - Algoritmo 2 **(b)** Progresión objeto c Escenario 2 - Algoritmo 2

Figura 5.8: Progresión objetos Escenario 2 - Algoritmo 2

Por otra parte, podemos ver como la tendencia sigue indicando cómo el número de $a's$ desciende hasta llegar a 0. Esto puede razonarse teniendo en cuenta la forma en la que el algoritmo calcula que regla ejecutar en cada paso. Si tomamos una configuración del sistema en el que tengamos objetos de cada tipo, es decir, tengamos $a's$, $b's$ y $c's$ entonces, la probabilidad que se asignaría a cada regla (teniendo en cuenta que la suma de las constantes es 1.2) sería la siguiente:

- Para la regla $r_1^1 : a \xrightarrow{0.2} aab$ tendremos una probabilidad $P(r_1^1) = 0.2/1.2 = 0.16667$

- Para la regla $r_2^1 : aa \xrightarrow{0.5} c$ tendremos una probabilidad $P(r_2^1) = 0.5/1.2 = 0.41667$
- Para la regla $r_3^1 : ac \xrightarrow{0.5} ab$ tendremos una probabilidad $P(r_3^1) = 0.5/1.2 = 0.41667$

Podemos ver como las dos últimas reglas son las más probables sumando aproximadamente un 0.83334 de la probabilidad de ejecución. Además, estás dos reglas suponen o bien dejar constante el número de $a's$ o bien eliminar dos. En consecuencia, todo parece indicar que la tendencia será que el número de $a's$ disminuya conforme avancen los pasos de computación y que, para un número suficiente de pasos llegue a cero. Por último, en este caso la tendencia con las $c's$ es una tendencia ascendente (Figura 5.8.b). Para ver cómo progresa el número de $c's$ tomamos los resultados de progresión de 5 experimentos elegidos al azar y los representamos frente a los pasos de computación para obtener la gráfica que aparece en la Figura 5.9.

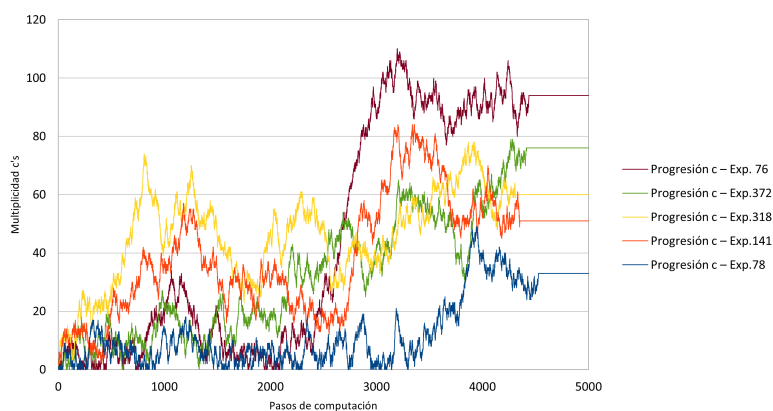


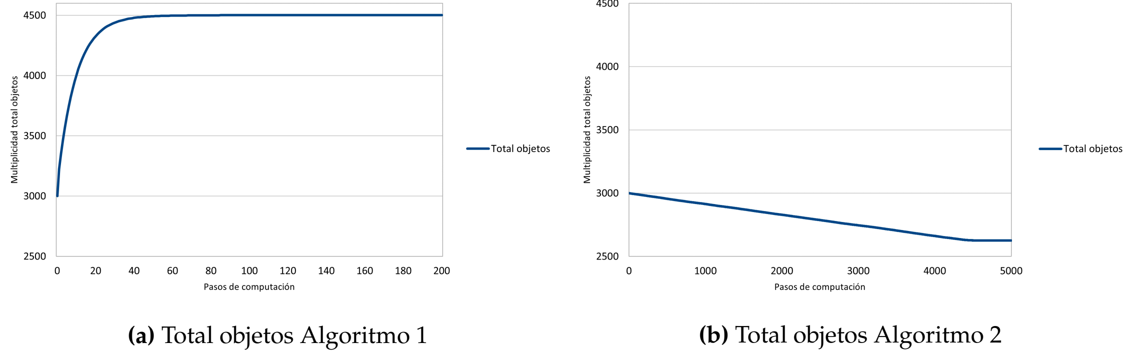
Figura 5.9: Progresión c's Escenario 2 - Algoritmo 2.

De las diferentes progresiones podemos ver por una parte la elevada cantidad de picos y valles que presentan las líneas de tendencia. Esto podría explicarse teniendo en cuenta que, al ejecutar una sola regla por paso, en algunas ejecuciones aumentaremos en 1 el número de $c's$ (regla r_2^1), en otras lo reduciremos en 1 (regla r_3^1) y en algunas permanecerá igual (regla r_1^1). Por tanto, el leve crecimiento que puede verse en estas gráficas lo podríamos atribuir a la propia aleatoriedad en la elección de las reglas.

5.2.3. Comparación Algoritmo 1 - Algoritmo 2

Para comparar ambos algoritmos en este escenario podemos tener en cuenta varias aspectos que ya hemos comentado. Vemos que ambos algoritmos se comportan de tal forma que número de $a's$ disminuye hasta que eventualmente llega a cero. Además, ambos tienden también a que el número de $b's$ crezca conforme avanzan los pasos por el motivo que hemos expuesto ya previamente. En cuanto a las $c's$ hemos visto que la tendencia en el Algoritmo 1 indicaba una disminución en su multiplicidad pues la regla que las genera terminaba por ejecutarse un número menor de veces que la regla que las consumía. En el Algoritmo 2 sin embargo, al hacer un reparto basado en probabilidad se observaba una mayor variabilidad en el número de $c's$ que tendían a ascender.

Otro aspecto a destacar es que, mediante el Algoritmo 1, la tendencia con respecto al total de objetos es que aumente con cada paso. Sin embargo, en el Algoritmo 2 no ocurre lo mismo y la tendencia es que el total de objetos se reduzca conforme avanzan los pasos. Estas tendencias se mantienen hasta que eventualmente el escenario se estabiliza y pueden apreciarse en las gráficas que aparecen en la Figura 5.10 a continuación.



(a) Total objetos Algoritmo 1 **(b)** Total objetos Algoritmo 2

Figura 5.10: Multiplicidad total de objetos para cada algoritmo

Esta tendencia a reducir el número total de objetos puede explicarse teniendo en cuenta las probabilidades que vimos en el subapartado 5.2.2. Es decir, las reglas r_2^1 y r_3^1 suman una probabilidad aproximadamente del 0.83334 y, estas reglas suponen o bien disminuir el número total de objetos o bien dejarlo igual. Por otra parte, la regla r_1^1 es la única que supone aumentar el número total de objetos pero, como vimos también, es la más improbable que se ejecute teniendo una probabilidad del 0.16667. En consecuencia, es de esperar que la tendencia para un número suficiente de pasos sea que el número total de objetos disminuya. En cuanto al Algoritmo 1, siempre ejecutaremos la regla r_1^1 cierto número de veces en cada paso de computación, lo que supondrá aumentar el número de objetos en la membrana (aunque esta regla se ejecute un número menor de veces que r_2^1 genera dos objetos nuevos por lo que siempre se crearán más objetos de los que se consumen).

En conclusión, podemos afirmar que ambos algoritmos presentan la misma tendencia de consumir todas las $a's$ y aumentar el número de $b's$ pero que existen diferencias significativas en el número total de objetos que se generan durante el proceso de ejecución así como en la multiplicidad de las $c's$ y por tanto, presentan comportamientos ligeramente distintos.

5.3 Escenario 3

Procedemos a analizar el Escenario 3 comenzando por una descripción formal del mismo.

$$\Pi = (O, L, \mu, \mathcal{M}_1, \mathcal{M}_2, R_1, R_2)$$

dónde:

- $O = \{a, b, c, d\}$
- $L = \{1, 2\}$
- $\mu = [_1 [_2]_2]_1$
- $\mathcal{M}_1 = (1, a^{30})$
- $\mathcal{M}_2 = (2, a^{10} b^{13})$
- $R_1 = \{r_1^1 : ab \xrightarrow{1.0,0.2} aab_{out}, r_2^1 : ac \xrightarrow{1.0,0.5} cc_{out}, r_3^1 : aa \xrightarrow{1.0,1.0} abc\}$

- $R_2 = \{r_1^2 : ac \xrightarrow{1.0,0.2} ad_{in_1}aa_{in_1}b_{out}, r_2^2 : bbbc \xrightarrow{1.0,0.2} e_{in_1}a_{out}\}$

Una representación visual mediante diagrama de Venn es la que puede verse en la Figura 5.11 a continuación. De la definición formal y el diagrama se desprende que nos encontramos frente a un sistema P con dos membranas, con multiconjuntos iniciales a^{30} para la interna y $a^{10}b^{13}$ para la externa (membrana piel). Además, estas membranas cuentan con reglas que permiten mover objetos entre ellas e incluso al exterior del sistema. Por ejemplo, la regla r_1^1 permite mover una b entre de la membrana interna a la membrana piel y la regla r_2^2 permite mover una e desde la membrana piel a la interna y una a al exterior del sistema (recordemos que si un objeto "sale" al exterior del sistema ya no podrá interactuar con sus objetos durante el resto de los pasos de computación).

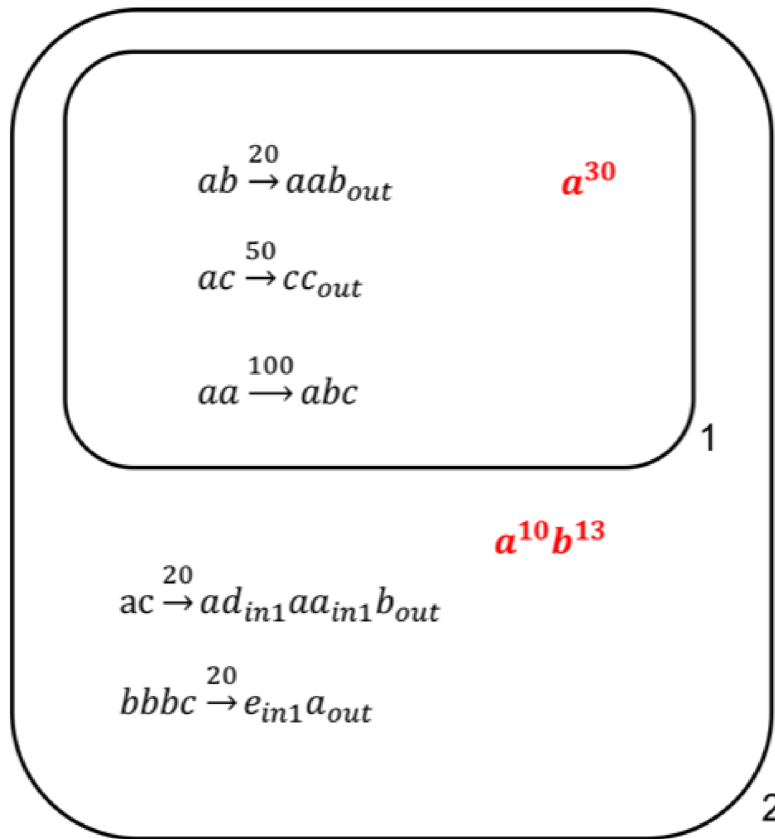
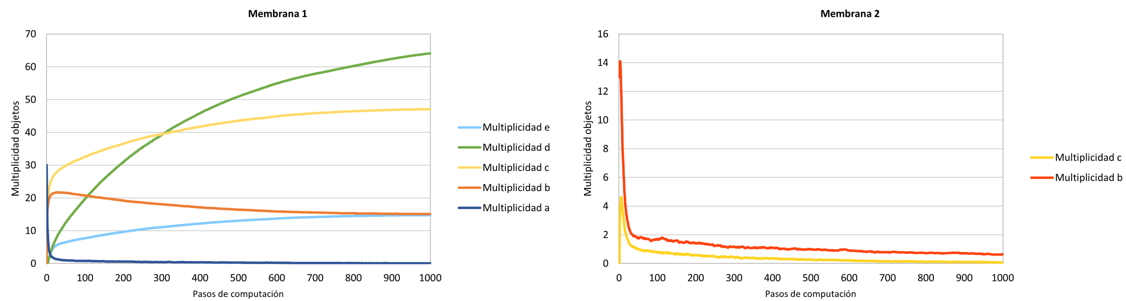


Figura 5.11: Escenario 3.

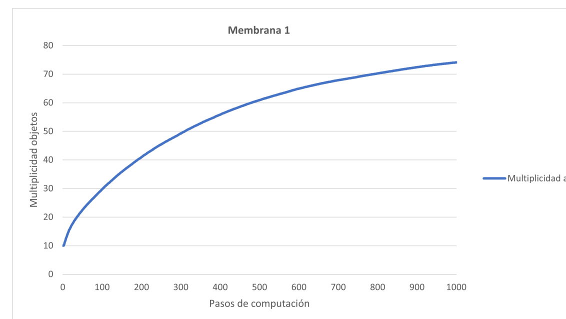
5.3.1. Escenario 3 - Algoritmo 1

El experimento que se lanzó con este escenario para el Algoritmo 1 consistió en un experimento de 500 ejecuciones con 1000 pasos de computación para cada una. La progresión de los diferentes objetos es la que puede verse en la Figura 5.12 a continuación. De estos resultados pueden resultar llamativos varios aspectos. Por una parte, en la membrana 1 tenemos una tendencia creciente de todos los objetos excepto la a y la b que, en principio, experimenta un crecimiento rápido durante los primeros pasos y luego decrece paulatinamente. De las partes izquierdas de las reglas en la membrana 1 puede verse cómo todas las reglas emplean $a's$ para ejecutarse y, en consecuencia, si las $a's$ quedan a cero, ninguna regla podrá ejecutarse hasta que eventualmente entre en la membrana alguna a generada por la ejecución de r_1^2 en la membrana 2.



(a) Progresión objetos membrana 1 Escenario 3 - Algoritmo 1.

(b) Progresión objetos b y c membrana 2 Escenario 3 - Algoritmo 1.



(c) Progresión objeto a membrana 2 Escenario 3 - Algoritmo 1.

Figura 5.12: Progresión objetos Escenario 3 - Algoritmo 1.

Vamos a tratar de explicar en primer lugar las tendencias crecientes de los objetos en la membrana 1. Por una parte, los objetos d y e sólo pueden aparecer en la membrana 1 a partir de la ejecución de la regla que los genera en la membrana 2. Además, estos objetos no aparecen en la parte izquierda de ninguna de las reglas de la membrana 1. Por tanto, la única opción posible es que la multiplicidad de estos objetos crezca de forma monótona. Por otra en primer lugar, la regla r_2^1 es la única que consume $c's$ pero, a cambio, genera una c que se queda en la membrana 1 y otra que pasa a la membrana 2. En segundo lugar, la regla r_3^1 no consume ninguna c pero sí que genera una. Por todo esto, el número de $c's$ en la membrana 1 únicamente puede crecer o, en algunos casos, permanecer constante.

Por otra parte, para entender el motivo del descenso en el número de $a's$ vamos a hacer una traza del primer paso de computación en la membrana 1. En el primer paso de computación únicamente tenemos $a's$ y, por tanto, únicamente puede ejecutarse una regla de la siguiente manera.

- Para la regla $r_3^1 : aa \xrightarrow{1.0} abc$ tendremos $\frac{1}{1} \cdot \frac{1}{2} \cdot 30 \cdot 1.0 = 15$ ejecuciones

Por tanto, al final del primer paso de computación tendremos 15 $a's$, 15 $b's$ y 15 $c's$ por lo que podrán pasar a ejecutarse todas las reglas de la membrana. De estas tres reglas, vemos que la primera (r_1^1) supone aumentar en uno el número de $a's$ mientras que la segunda (r_2^1) y la tercera (r_3^1) suponen disminuirla en uno. Además, estas dos reglas cuentan con un valor más alto de la constante estocástica que la primera y, por tanto, se deben ejecutar un número mayor de veces. En consecuencia, para cierto número de pasos la multiplicidad de a debería aproximarse a cero. Un comportamiento peculiar de este sistema es que en la membrana 1 las $a's$ quedan cercanas o iguales a cero pero, por el hecho de que la membrana 2 puede introducir nuevas $a's$ en la membrana 1 se produce

una oscilación de la multiplicidad entre diferentes valores cercanos a cero. Para verlo, podemos tomar la progresión de las $a's$ en tres ejecuciones tomadas al azar y representarlas frente a los pasos de computación. Dicha gráfica es la que puede verse en la Figura 5.13 a continuación.

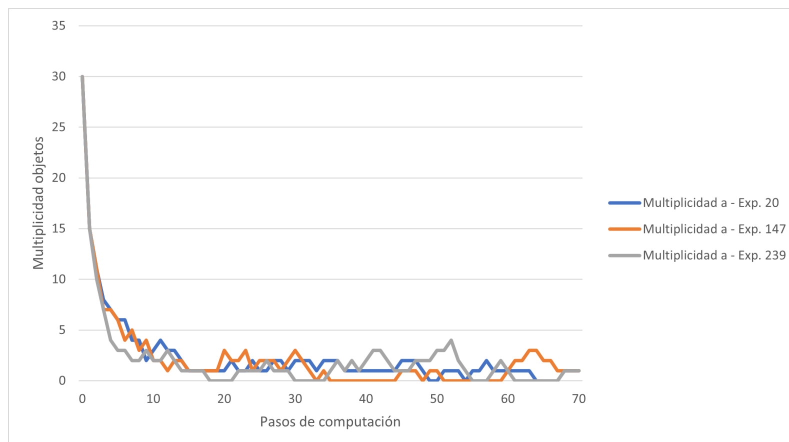


Figura 5.13: Progresión a's Escenario 3 - Algoritmo 1.

De esta gráfica puede verse cómo el número de $a's$ oscila en valores cercanos al cero y a pesar de tomar valores exactamente cero durante algunos pasos (y por tanto detener la ejecución de reglas en la membrana 1) nada impide que vuelvan a entrar $a's$ desde la segunda membrana y se reanude la ejecución de reglas.

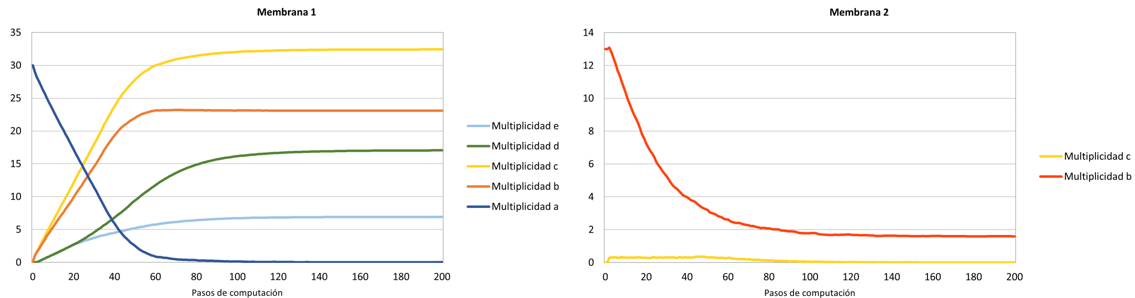
Por último, en cuanto a la tendencia de las $b's$ vemos que tiene un pico al inicio para seguidamente descender lentamente conforme avanzan los pasos de computación. Si nos fijamos en la Figura 5.12.a podemos ver que el descenso de las $b's$ se produce cuando la multiplicidad de a alcanza números cercanos al cero. Cuando a valga uno, la regla r_3^1 no podrá ejecutarse por lo que no se podrán generar nuevas $b's$. En consecuencia, la ejecución de r_1^1 (que sí que será aplicable pues tenemos suficientes $a's$ para ello) supondrá disminuir su multiplicidad en uno. Por esto mismo, la multiplicidad de b tenderá a disminuir con repuntes puntuales cuando entre una (o más) $a's$ desde 2 y se pueda volver a aplicar r_3^1.

Pasamos a continuación a examinar la progresión de los objetos en la membrana 2. Por una parte vemos como la multiplicidad de las $a's$ experimenta un crecimiento monótono, esto es debido a que, la regla r_1^2 consume una única a pero a cambio genera dos nuevas y, por tanto, mientras tengamos $c's$ disponibles en la membrana será de esperar que la multiplicidad de a crezca (además ninguna otra regla consume $a's$ en su parte izquierda). En lo que respecta a la progresión de las $b's$ se percibe un crecimiento durante los primeros pasos de computación que tiende a invertirse y convertirse en descenso a partir del paso 4. Esto es así porque, durante estos primeros pasos la regla r_1^1 emite a la membrana 2 un número mayor de $b's$ de las que se consumen con la regla r_2^2. Como hemos visto, llega cierto punto de la ejecución en el que el número de $a's$ en la membrana 1 oscila entorno al cero y, por tanto, es de esperar que se transfiera un número menor de $b's$ a la membrana 2. Llegado a este punto, la regla r_2^2 comenzará a consumir rápidamente tanto las $b's$ como las $c's$. Por ese mismo motivo, el descenso en la multiplicidad de estos dos últimos objetos se da aproximadamente en el mismo paso.

5.3.2. Escenario 3 - Algoritmo 2

El experimento que se lanzó con este escenario para el Algoritmo 2 consistió en un experimento de 500 ejecuciones con 5000 pasos por ejecución. Sin embargo, ninguna eje-

cución logro superar el paso 198 con un número de $a's$ superior a uno en la membrana 1 y un número de $c's$ superior a uno en la membrana 2. Recordemos que la a aparecía en la parte izquierda de todas las reglas en la membrana 1 y la c en la de todas las reglas de la membrana 2 y, por este motivo, si estos dos objetos se agotan no se puede ejecutar ninguna regla adicional en este sistema. Las progresiones de los diferentes objetos, son las que aparecen en las gráficas de la Figura 5.14.



(a) Progresión objetos membrana 1 Escenario 3 - Algoritmo 2.

(b) Progresión objetos b y c membrana 2 Escenario 3 - Algoritmo 2.

(c) Progresión objeto a membrana 2 Escenario 3 - Algoritmo 2.

Figura 5.14: Progresión objetos Escenario 3 - Algoritmo 1.

Comenzamos de nuevo planteando un análisis de las progresiones en la membrana 1. En este caso observamos una tendencia creciente de todos los objetos excepto el objeto a que experimenta un descenso hasta eventualmente llegar a cero. De nuevo, la multiplicidad de los objetos d y e es monótona creciente pues entran desde la membrana 2 y no son consumidos por la parte izquierda de ninguna regla en la membrana 1. En cuanto a la multiplicidad de los objetos a, b y c podemos pararnos a examinar las probabilidades de cada regla cuando las tres son aplicables.

- Para la regla $r_1^1 : ab \xrightarrow{0.2} aab_{out}$ tenemos $P(r_1^1) = 0.2/1.7 = 0.117847$
- Para la regla $r_2^1 : ac \xrightarrow{0.5} cc_{out}$ tenemos $P(r_2^1) = 0.5/1.7 = 0.294118$
- Para la regla $r_3^1 : aa \xrightarrow{1.0} abc$ tenemos $P(r_3^1) = 1.0/1.7 = 0.588235$

De estas probabilidades vemos que las reglas r_2^1 y r_3^1 suman una probabilidad de 0.882353 y ambas suponen disminuir el número de $a's$ (en dos para r_3^1 y en uno para r_2^1), mientras que la regla r_1^1 supone aumentarlo en uno. En consecuencia, para un número suficiente de pasos, las $a's$ se agotarán llegando su multiplicidad a cero. Vemos además que la regla r_3^1 supone aumentar en uno la multiplicidad de b como de c (la regla r_2^1 supone dejar constante la multiplicidad de c) y por tanto, es de esperar que la multiplicidad

de estos dos objetos aumente conforme avanzan los pasos de computación. Se producirán descensos puntuales en el número de $b's$ al aplicarse r_1^1 pero, al ser la regla menos probable, no impedirá que la tendencia de este objeto sea creciente. Esto también explica porqué la multiplicidad de c consigue valores más altos que la de b.

Para terminar, en cuanto a la membrana 2 observamos que la tendencia es a agotar todas las $c's$ y por tanto impedir que se aplique ninguna regla adicional. Por una parte, vemos que la regla r_1^2 supone consumir una a y generar tres a cambio, dos que permanecen en la membrana 2 y una que entra a la membrana 1. Además, la regla r_2^2 no supone consumir ninguna a (genera una que pasa al exterior del sistema). Por tanto, la multiplicidad de las $a's$ crecerá conforme avanzan los pasos de computación. Por otra parte, la multiplicidad de b experimenta un crecimiento durante los primeros pasos debido a la aplicación de la regla r_1^1 pero rápidamente sufre un descenso debido a la aplicación de r_2^2 que supone consumir tres $b's$ sin generar ninguna a cambio. Recordemos que la regla r_1^1 era la que menos ejecuciones recibía en la membrana 1 pues era la menos probable, en consecuencia, en la membrana 2 se consumirán más $b's$ de las que entran desde 1. Por último, en cuanto a las $c's$ observamos que su multiplicidad oscila en todo momento entre cero y uno. Esto es debido a que este algoritmo ejecuta una única regla por paso de computación y, por tanto, las $c's$ entran en la membrana 2 una a una. Además, como la c aparece en las partes izquierdas de las reglas en la membrana 2 siempre que exista una disponible será consumida por alguna de estas reglas.

5.3.3. Comparación Algoritmo 1 - Algoritmo 2

Si bien ambos algoritmos presentan ciertas tendencias en común existe una diferencia clara. Esta diferencia consiste en la progresión de la multiplicidad de las $c's$ en la membrana 2. Como hemos visto en la explicación anterior, el hecho de que en el Algoritmo 2 ejecute una sola regla por bloque en cada paso de computación, hace que una única c pueda entrar en esta membrana que será inmediatamente consumida por alguna de sus reglas. Por otra parte, el Algoritmo 1 puede introducir un número de $c's$ mayor que uno en esta membrana durante cierto paso de computación, lo que permite la aplicación de un mayor número de reglas. Vemos por tanto que la forma de proceder de ambos algoritmos puede generar diferencias significativas cuando los sistemas cuentan con más de una membrana y reglas que suponen intercambiar objetos entre ellas.

Este comportamiento del Algoritmo 2 hace que se pierda cierta proporcionalidad en la ejecución de reglas entre bloques que sí proporciona el Algoritmo 1. Por ejemplo, con el Algoritmo 1, si tomamos una ejecución en la que la membrana 1 cuenta con el multiconjunto $a^{12}b^{16}c^{17}$ y la membrana 2 cuenta con el multiconjunto $a^{10}b^{14}c^{2}$ entonces podremos ejecutar siete reglas en la membrana 1 y una regla en la membrana 2 (mencionar que los números de ejecuciones obtenidos al aplicar el Algoritmo 1 resultan decimales, por tanto, el número final elegido podría variar de una ejecución a otra). En este caso vemos que tenemos una proporción de 7 reglas ejecutadas para el bloque de la membrana 1 por cada regla ejecutada en el bloque de la membrana 2. Esta proporcionalidad sin embargo siempre será de uno a uno si aplicamos el Algoritmo 2. Pensamos que esta diferencia en las proporcionalidades entre bloques de reglas marca una diferencia significativa entre los dos algoritmos implementados y como veremos en la sección 5.5, esta diferencia generará ciertos problemas a la hora de aplicar el Algoritmo 2 al escenario del virus COVID-19.

5.4 Escenario 4

Procedemos con el Escenario 4 proponiendo la siguiente definición formal del mismo:

$$\Pi = (O, L, \mu, \mathcal{M}_1, R_1)$$

dónde:

- $O = \{a, b, c, d, e, f\}$
- $L = \{1\}$
- $\mu = [_1]_1$
- $\mathcal{M}_1 = (1, a^{100}b^{100}c^{100})$
- $R_1 = \{r_1^1 : ab \xrightarrow{1.0,0.2} acd, r_2^1 : bc \xrightarrow{1.0,0.5} bce, r_3^1 : cc \xrightarrow{1.0,0.6} bf\}$

Una representación visual de este escenario es la que puede verse en el diagrama de Venn que aparece en la Figura 5.15. Como se desprende de la definición formal y el diagrama nos encontramos frente a un sistema P con una única membrana, un multiconjunto inicial $a^{100}b^{100}c^{100}$, y tres reglas que lo pueden hacer evolucionar. De este sistema, resulta interesante destacar las partes derechas de las reglas. Podemos ver que una de las reglas genera una d en su parte derecha, otra de ellas genera una e y la última genera una f. El hecho de que estos objetos se generen a partir de una única regla y que además no aparezcan en ninguna parte izquierda nos permite de forma indirecta llevar un conteo de cuántas veces se ha ejecutado cada una. Aprovecharemos esta particularidad que presenta este sistema para hacer un análisis en profundidad de cómo funcionan los algoritmos en este caso concreto.

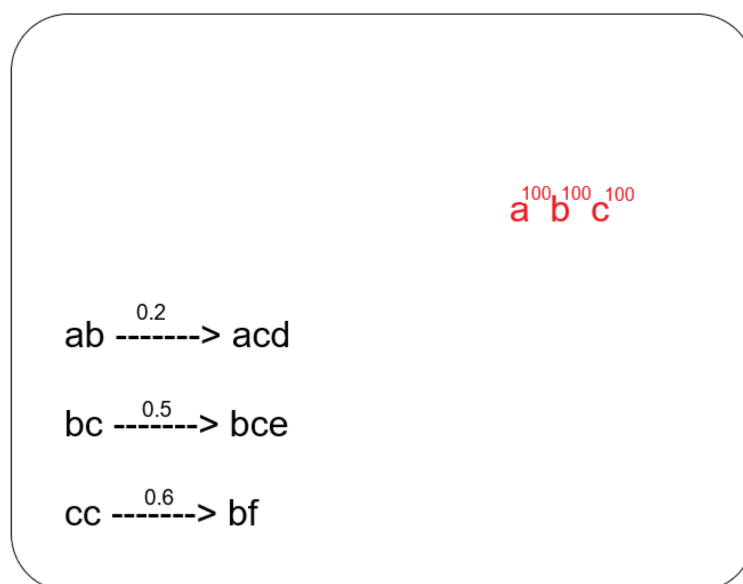
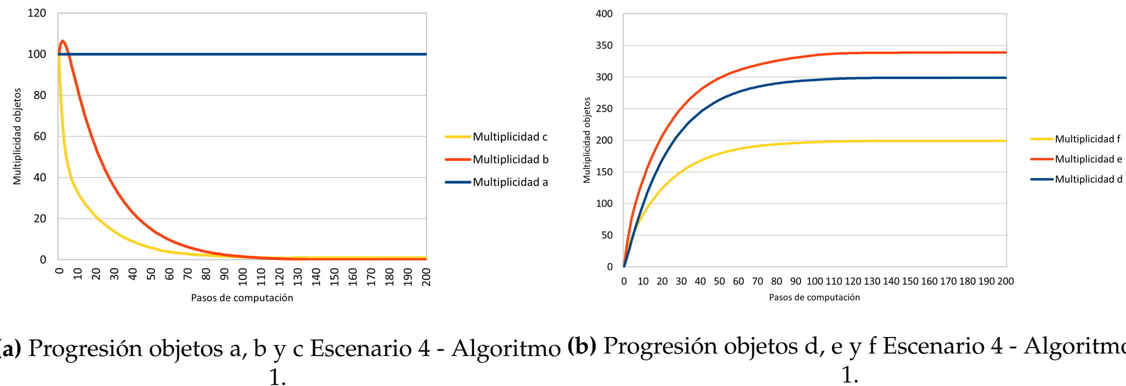


Figura 5.15: Escenario 4

5.4.1. Escenario 4 - Algoritmo 1

El experimento que se lanzó con este escenario para el Algoritmo 1 consistió en un experimento de 500 ejecuciones con 1000 pasos de computación para cada una. La progresión de los objetos puede verse en la Figura 5.16 a continuación. Para mayor claridad y facilidad a la hora de visualizar e interpretar los datos, hemos dividido las progresiones en dos gráficas.



(a) Progresión objetos a, b y c Escenario 4 - Algoritmo 1.

(b) Progresión objetos d, e y f Escenario 4 - Algoritmo 1.

Figura 5.16: Progresión objetos Escenario 4 - Algoritmo 1.

Debemos mencionar que en las gráficas de las progresiones únicamente hemos incluido hasta el paso 200 en el eje de abscisas. Esto es así porque ninguna ejecución logró superar el paso 172 con un número de $b's$ superior a cero y un número de $c's$ superior a uno. Si tenemos cero $b's$ entonces ninguna de las primeras dos reglas pueden aplicarse y, si tenemos una única c entonces la última regla tampoco puede aplicarse porque requiere dos $c's$ en su parte izquierda.

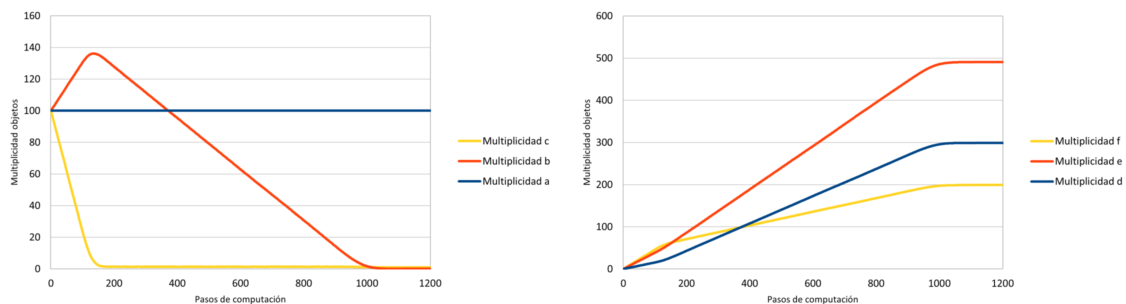
De las progresiones obtenidas, puede resultar llamativo en primer lugar que el objeto a mantenga su multiplicidad constante a 100 durante todos los pasos de computación. Esto se puede explicar contemplando la regla r_1^1 que es la única que consume $a's$ en su parte izquierda y la única que las genera en su parte derecha. En concreto, como únicamente consume una a y genera a cambio otra a la multiplicidad de este objeto no variará independientemente de cuántas veces se ejecute la regla en cuestión. Por otra parte, se puede observar cómo los objetos b y c presentan una tendencia a reducir su multiplicidad hasta que, eventualmente, b llega a cero y c llega a uno.

En principio, la multiplicidad de b sufre un pequeño crecimiento puesto que, en los dos primeros pasos se generan más $b's$ de las que se consumen, mientras que, por los valores de las constantes estocásticas, el valor de las $c's$ disminuye. De la Figura 5.16.b podemos ver como la multiplicidad de d resulta ser mayor que la de f, en concreto, al terminar la simulación disponemos de 300 $d's$ y 200 $f's$. Esto indica que la regla r_1^1 se ha ejecutado 300 veces y la regla r_3^1 200 veces. Teniendo en cuenta que la regla r_3^1 consume 2 $c's$ y la regla r_1^1 únicamente genera una, entonces se en cada paso de computación se consumirán mas $c's$ de las que se generan, lo que explicaría su descenso hasta 1. El mismo razonamiento se puede aplicar a b la regla r_1^1 supone consumir una y la regla r_3^1 generar una (la regla r_2^1 supone no alterar la multiplicidad) y por tanto, en cada paso de computación se consumirán más $b's$ de las que se generan.

Por último, los objetos d, e y f sufren una tendencia monótona creciente. El motivo de este comportamiento es el que explicamos al principio de esta sección, es decir, estos objetos únicamente aparecen en las partes de derechas de las reglas y, en consecuencia, no existe la posibilidad de que sean consumidos y su multiplicidad disminuya.

5.4.2. Escenario 4 - Algoritmo 2

Pasamos a analizar los resultados obtenidos mediante el Algoritmo 2, en este caso se lanzó un experimento de 500 ejecuciones con 2000 pasos por ejecución. Este esquema se lanzó teniendo en cuenta que mediante el Algoritmo 1 se generaron un total de 840 objetos nuevos aproximadamente y se buscaba tratar de alcanzar al menos la misma multiplicidad total mediante el segundo algoritmo. Las progresiones obtenidas son las que pueden verse en la Figura 5.17 a continuación. Mencionar además que, ninguna ejecución logró superar los 1112 pasos con un número de $b's$ superior a cero y un número de $c's$ superior a uno y por tanto únicamente representamos hasta el paso 1200 en el eje de abscisas.



(a) Progresión objetos a, b y c Escenario 4 - Algoritmo 2. **(b)** Progresión objetos d, e y f Escenario 4 - Algoritmo 2.

Figura 5.17: Progresión objetos Escenario 4 - Algoritmo 2.

Para este algoritmo vemos exactamente las mismas tendencias que con el Algoritmo 1, es decir, las $a's$ se mantienen con una multiplicidad de 100 (por los mismos motivos que expusimos en el subapartado 5.4.1) y las $b's$ y las $c's$ tienden a decrecer hasta llegar a 0 y 1 respectivamente. En cuanto a los objetos d, e y f estos presentan tendencias monótonas crecientes. Vemos que la multiplicidad de los objetos b y c presentan tendencia a decrecer aunque, de nuevo, la multiplicidad de b presenta un máximo y a partir de ese punto comienza a decrecer. Si observamos detenidamente la Figura 5.16.a podemos ver como este máximo coincide con el punto dónde la multiplicidad de las $c's$ alcanza un valor muy cercano o igual a 1. Si tratamos de replicar los cálculos que realiza este algoritmo para obtener las probabilidades de cada regla, obtendríamos los valores que siguen.

- Para la regla $r_1^1 : ab \xrightarrow{0.2} acd$ tendríamos una probabilidad $P(r_1^1) = 0.2/1.3 = 0.153846$
- Para la regla $r_2^1 : bc \xrightarrow{0.5} bce$ tendríamos una probabilidad $P(r_2^1) = 0.5/1.3 = 0.384615$
- Para la regla $r_3^1 : cc \xrightarrow{0.6} bf$ tendríamos una probabilidad $P(r_3^1) = 0.6/1.3 = 0.461538$

De estas probabilidades podemos ver que las dos últimas reglas suman aproximadamente una probabilidad del 0.846153 y que además esas dos reglas suponen o bien generar una b nueva o bien no alterar la multiplicidad de la misma. Por tanto, la multiplicidad de b crecerá conforme avancen los pasos de computación (con algún descenso puntual al ejecutarse la primera regla y consumir una b). Además, estas dos mismas reglas suponen, o bien reducir en 2 la multiplicidad de las $c's$ o bien dejarla igual, por tanto, también se producirá un descenso en el número de $c's$ conforme avancen los pasos (de nuevo, con algún incremento puntual al ejecutarse la primera regla).

Esta tendencia avanza hasta que, eventualmente, el número de $c's$ llega a uno y la tercera regla no puede seguir ejecutándose. En consecuencia, el algoritmo calculará las siguientes probabilidades para las dos reglas que sí pueden ejecutarse.

- Para la regla $r_1^1 : ab \xrightarrow{0.2} acd$ tendríamos una probabilidad $P(r_1^1) = 0.2/0.7 = 0.285714$
- Para la regla $r_2^1 : bc \xrightarrow{0.5} bce$ tendríamos una probabilidad $P(r_2^1) = 0.5/0.7 = 0.714286$

Lo que ocurrirá a partir de este punto será que, en la mayoría de ocasiones (más concretamente en un 71.4286 % de las ocasiones) se ejecutará las segunda regla y por tanto, el número de $b's$ y de $c's$ permanecerá constante generando nuevas $e's$. Eventualmente, la primera regla se ejecutará consumiendo una b y generando a cambio una nueva c que permitirá ejecutar de nuevo la regla r_3^1 y volviendo a barajar las probabilidades que exponíamos más arriba. De todo esto, se esperaría que, para un número suficiente de pasos la primera regla termine por ejecutarse hasta eventualmente consumir todas las $b's$ de la membrana. Esto puede verse en la Figura 5.17.b cuando se produce un cruce entre las multiplicidades de d y f pasando la multiplicidad de d a a ser superior a la de f lo que efectivamente indicaría que la primera regla r_1^1 comienza a ejecutarse un número mayor de veces que la tercera r_3^1.

Para terminar, como ya se ha comentado la multiplicidad de los objetos d, e y f presenta una tendencia monótona creciente por los mismos motivos que se explicaron en el subapartado 5.4.1. Este crecimiento se detiene en el paso 1112 pues ya no quedan $b's$ disponibles y únicamente resta una c por lo que no se pueden seguir ejecutando más reglas.

5.4.3. Comparación Algoritmo 1 - Algoritmo 2

Por todo lo visto hasta el momento, se puede concluir que ambos algoritmos presentan la mismas tendencias para todos sus objetos y ambos presentan la tendencia de consumir todas las $b's$ aunque, el Algoritmo 2 toma un mayor número de pasos para conseguirlo. Por tanto, para este escenario podemos concluir que los dos algoritmos presentan el mismo comportamiento y resultados muy similares.

5.5 Escenario COVID-19

Procedemos a acometer el escenario que simula la expansión del virus COVID-19 en un entorno que contempla 500 unidades familiares. La ejecución y obtención de resultados en este escenario consistía uno de los objetivos de este trabajo de fin de grado además de una forma de medir las capacidades del simulador así cómo el funcionamiento de los algoritmos en escenarios con un número elevado de membranas y objetos (elevado en comparación con los cuatro escenarios vistos en este capítulo). Vamos a comenzar haciendo una descripción de los objetos y las membranas que conformaban este sistema. De nuevo, debido al elevado número de objetos y membranas, en esta ocasión vamos a omitir la descripción formal pues consideramos que no aporta claridad a la explicación.

Comenzamos haciendo una descripción de los objetos que contenía este sistema. Por una parte, tenemos un tipo de objeto que representa a los habitantes del sistema y que está dividido en cuatro “segmentos”, es decir, adopta la forma $a_b_c_d$ dónde cada segmento tiene los siguientes significados.

- Primer segmento a: siempre toma el valor h indicando que el objeto representa a un habitante del sistema.
- Segundo segmento b: representa el rol de dicho habitante y puede tomar los valores:
  - t el habitante es un trabajador.
  - c el habitante cuida la casa.
  - e el habitante es un estudiante.
- Tercer segmento c: representa el estado de la infección en el habitante y puede tomar los valores:
  - s el habitante está sano.
  - i el habitante está infectado con el virus.
  - in el habitante está inmunizado contra el virus.
- Cuarto segmento d: representa el estado de salud del habitante y puede tomar los valores:
  - $e1$ el habitante es asintomático.
  - $e2$ el habitante presenta síntomas leves.
  - $e3$ el habitante está en estado grave.

Así, por ejemplo, un objeto $h_t_s_e1$ representaría a un habitante que trabaja y no tiene el virus (de ahí que sea asintomático), un objeto $h_e_i_e1$ representaría a un habitante que estudia y que tiene el virus pero es asintomático y un objeto $h_c_i_e2$ representaría a un habitante que cuida de la casa, tiene el virus y se encuentra en estado leve. Como explicábamos al inicio del capítulo, el escenario contempla un total de 500 unidades familiares. Cada unidad familiar está formada por un habitante que trabaja, otro que cuida de la casa y dos que estudian. En consecuencia, al inicio de la simulación tendremos 500 personas que trabajan, 500 que cuidan de la casa y 1000 que estudian. Además, para observar cómo avanza la infección, la simulación comienza con un trabajador infectado, es decir, la configuración inicial del sistema contiene un $h_t_i_e1$ que podrá contagiar a otros habitantes.

El escenario funciona de tal forma que, cada paso de computación transcurre una hora y, conforme van pasando las horas los habitantes se van moviendo entre las diferentes zonas (membranas) que constituyen el escenario. Resulta también importante comentar la existencia de ciertos objetos $horaYY$ siendo YY un entero entre 0 y 23 y que permite llevar un control de la hora del día en la que se encuentra el escenario. Este tipo de objetos nos permite además controlar a que horas se desplazan los habitantes entre zonas, por ejemplo, reglas cómo $h_c_s_e1\ hora8 \xrightarrow{1.0} hora9\ h_c_s_e1_{zonacomun}$ permite a los cuidadores de las casas desplazarse a la $zonacomun$ cuando llegan las 8 de la mañana. En cuanto a las zonas del escenario, vamos a explicar brevemente cuales son las que podemos encontrar.

- $casaXXX$: dónde XXX es un entero entre 1 y 500. Representan las casas dónde conviven las diferentes unidades familiares. Las personas que constituyen una familia pasan varias horas al día en una casa.
- $zonatrabajo$: es la zona a la que acuden los trabajadores cuando empieza su jornada.
- $zonaescuela$: es la zona a la que acuden los estudiantes cuando empiezan las clases.

- *zonacomun*: es la zona a la que acuden los habitantes de cualquier rol durante ciertas horas al día. Existen solapes entre las horas a las que acuden los diferentes roles y, por tanto, podemos tener situaciones en la que todos los habitantes estén compartiendo el mismo espacio.

- *zonahospital*: es la zona a la que acuden los habitantes cuando comienzan a padecer síntomas del virus.

De todo esto se desprende que el sistema está conformado por 505 membranas, una membrana *eco* que constituye la piel del sistema, 500 membranas del tipo $casaXXX$, una $zonacomun$, una $zonatrabajo$, una $zonaescuela$ y una $zonahospital$. Una representación visual simplificada de este escenario (sin incluir objetos) es la que puede verse en el diagrama de Venn en la Figura 5.18.

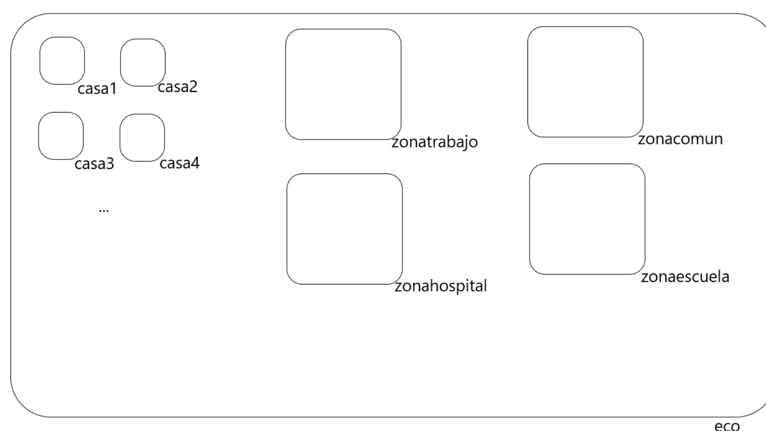


Figura 5.18: Escenario COVID-19

Este sistema, además, cuenta con un número elevado de reglas que determinan cómo avanza la infección por el escenario. Sin entrar en el detalle de describir todas ellas, vamos a tratar de exponer como las reglas regulan el movimiento de los habitantes en el escenario y de cómo se producen los contagios y las curas.

- Los trabajadores acuden al trabajo, los estudiantes al centro de estudios y los cuidadores a la zona común a hacer recados durante un intervalo horario.

- Si un habitante está asintomático continuará con su rutina diaria.

- Si un habitante desarrolla síntomas leves se quedará en casa.

- Si una persona pasa a estado grave será trasladado al hospital.

- Si hay dos habitantes se encuentran en la misma zona y una está contagiada existe una probabilidad de que una contagie a la otra.

- Un habitante infectado puede empeorar, es decir, puede pasar de asintomático a leve y de leve a grave.

- Un habitante en estado grave puede fallecer.

- Si un habitante se cura puede volver a su rutina habitual.

- Un habitante que se cura puede desarrollar inmunidad al virus lo que impediría que se pudiese volver a contagiar.

- Un habitante puede contagiarse en cualquier zona del escenario.

Para obtener las estadísticas de la ejecución de este escenario utilizamos el método 2 que se explicó en el apartado 3.4, es decir, mostramos la progresión de ciertos objetos seleccionados en ciertas membranas también previamente seleccionadas. En concreto, llevamos un conteo de los objetos que representan a habitantes en las membranas $zonacomun$, $zonatrabajo$, $zonaescuela$ y $zonahospital$. Esto nos permitía llevar un registro de cuánta gente acudía al trabajo, a estudiar, a hacer recados a la zona común y cuánta gente estaba hospitalizada en cada momento. Además, llevamos un recuento de estos mismos objetos en la membrana piel *eco*, lo que nos permitía llevar un conteo general de cuánta gente había en cada estado en cada paso de la simulación.

En cuanto a la fase de experimentación, antes de proseguir cabe destacar ciertos aspectos y problemas que surgieron durante el proceso. En primer lugar, después de lanzar varias ejecuciones con el Algoritmo 2 entendimos que este algoritmo no resultaba adecuado para las necesidades de ejecución de este escenario. Por una parte, nos dimos cuenta de que el algoritmo eliminaba al único infectado con el virus en el primer paso de computación, lo que impedía que éste se expandiese entre la población y por tanto observar cómo progresaban las infecciones y las curas. Esto es así debido a que durante la ejecución se generaba el siguiente bloque de reglas.

$$BR = \{h_t_i_e1 \xrightarrow{0.002} h_t_in_e1, h_t_i_e1 \xrightarrow{0.001} h_t_s_e1, h_t_i_e1 \xrightarrow{0.0005} h_t_i_e2\}$$

Si obtenemos las probabilidades de acuerdo al algoritmo 2 sabiendo que la multiplicidad de $h_t_i_e1$ es uno, entonces tendremos lo siguiente:

- Para la regla $h_t_i_e1 \xrightarrow{0.002} h_t_in_e1$ tenemos 0.002 / 0.0035 = 0.571429
- Para la regla $h_t_i_e1 \xrightarrow{0.001} h_t_s_e1$ tenemos 0.001 / 0.0035 = 0.285714
- Para la regla $h_t_i_e1 \xrightarrow{0.0005} h_t_i_e2$ tenemos 0.0005 / 0.0035 = 0.142857

Podemos ver como las dos primeras reglas suman una probabilidad del 0.857143 y suponen o bien curar el enfermo y que quede sano, o bien curarlo y que desarrolle inmunidad al virus. En consecuencia será muy probable que en el primer paso el enfermo se cure y que la infección no pueda extenderse. Este problema no afecta al Algoritmo 1 pues ocurriría lo siguiente.

- Para la regla $h_t_i_e1 \xrightarrow{0.002} h_t_in_e1$ tenemos $\frac{1}{3} \cdot 1 \cdot 0.002 = 0.00066$ ejecuciones.
- Para la regla $h_t_i_e1 \xrightarrow{0.001} h_t_s_e1$ tenemos $\frac{1}{3} \cdot 1 \cdot 0.001 = 0.00033$ ejecuciones.
- Para la regla $h_t_i_e1 \xrightarrow{0.0005} h_t_i_e2$ tenemos $\frac{1}{3} \cdot 1 \cdot 0.0005 = 0.000166$ ejecuciones.

Al ser el número de ejecuciones tan cercano a cero la probabilidad de que alguna de estas tres reglas se ejecute una vez será muy baja (concretamente, tan baja cómo el número de ejecuciones que hemos obtenido). Existen otros motivos por los que el Algoritmo 2 no resulta adecuado para este escenario. En algunas circunstancias necesitamos que ciertos bloques se ejecuten de forma maximal, por ejemplo, necesitamos que todos los objetos *horaXX* cambien en cada paso para controlar la hora, sin embargo, mediante este algoritmo no podemos garantizar que esto ocurra. Otro motivo es que, como vimos en

la sección 5.3.3 al comparar algoritmos para el escenario 3, el algoritmo 2 no respetaba las proporciones de reglas ejecutadas en diferentes bloques (como sí hacía el algoritmo 1) lo que ocasionaría una evolución no deseada del escenario. Vistos los aspectos más relevantes de este escenario, procedemos a hacer una análisis de los resultados obtenidos mediante el Algoritmo 1.

5.5.1. Escenario COVID-19 - Algoritmo 1

Para la experimentación de este escenario con el Algoritmo 1, se escogió un esquema de una ejecución con 15000 pasos de computación. Este esquema se escogió de esta forma porque, en este caso no resultaba de especial interés obtener las medias de un número elevado de ejecuciones, sino lanzarlas por separado y analizar los resultados de cada una de ellas. Procedemos por tanto a analizar los resultados obtenidos con una de las ejecuciones lanzadas. Mencionar además que, simular 15000 pasos en este escenario resulta equivalente a simular 625 días completos. El objetivo fue conseguir "estabilizar" el escenario, es decir, hacer evolucionar la población hasta conseguir una mayoría inmunizada.

En primer lugar, vamos a hacer un análisis de la evolución de la población en general, es decir, vamos a centrarnos en el total de sanos, inmunizados, infectados y fallecidos, sin distinguir entre los roles de las personas que conforman cada grupo. Los resultados de estas progresiones pueden verse en la Figura 5.19 a continuación. De estas progresiones recordar que partimos de 1999 personas sanas y una persona infectada. Se puede observar cómo, durante aproximadamente los primeros 3000 pasos de computación se produce una evolución significativa de los cuatro grupos que hemos diferenciado. Por una parte, tenemos un descenso de la población sana junto con un rápido incremento de la población de infectados. Concretamente, la población de infectados evoluciona hasta alcanzar su máximo de 1297 infectados en el paso 1974 (es decir 82 días y 6 horas después de comenzar la simulación).

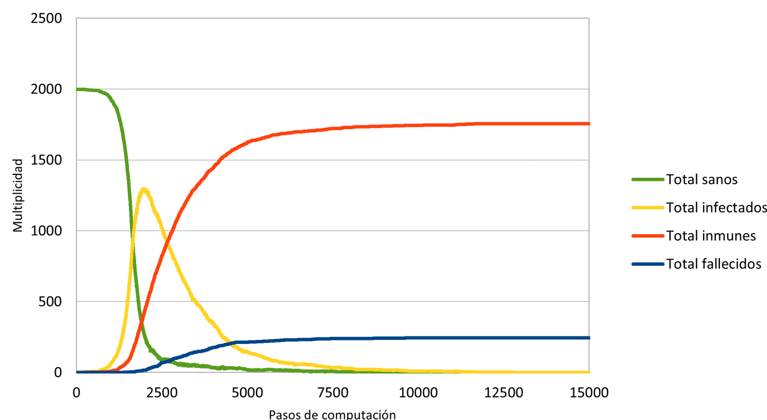


Figura 5.19: Progresión sanos, infectados, inmunes y fallecidos escenario COVID-19.

A partir de este paso, se produce un descenso de la población de infectados acompañada por un aumento de la población de inmunizados. Al concluir el experimento, la población se divide en cero personas infectadas, cero personas sanas, 1755 personas inmunizadas y 245 personas fallecidas. Para ver más en detalle cómo evoluciona la infección, resulta interesante hacer una comparación de cómo evolucionan los infectados en función de los roles que desempeñan. Los resultados de estas progresiones pueden verse en la gráfica de la Figura 5.20. Por una parte, vemos que el pico más elevado de infecciones corresponde a la población de estudiantes con un máximo de 662 infectados. Por otra parte, el máximo de infectados correspondiente a la población de trabajadores es de 351

infectados y, por último, un máximo de 300 infectados para la población de cuidadores de la casa.

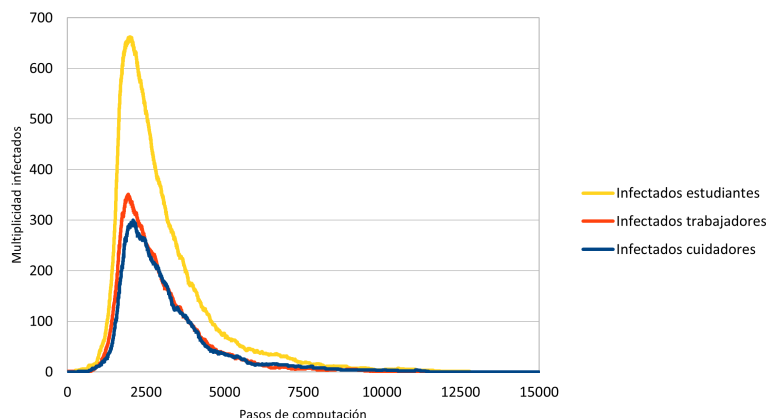


Figura 5.20: Progresión infectados por rol escenario COVID-19

De estos datos se puede concluir que, contra más numeroso es un grupo de la población más contagios se deberían producir en el mismo. Se observa sin embargo cierta diferencia en el máximo de infectados entre la población de trabajadores y la de cuidadores a pesar de contar con el mismo número inicial de individuos. Esta diferencia puede atribuirse al hecho de que los trabajadores pasan un mayor número de horas fuera de casa y por tanto es más probable que se contagien. Concretamente, los trabajadores pasan seis horas al día en su puesto de trabajo (y por tanto, estando en contacto con otros trabajadores que podrían estar infectados) y dos horas en la zona común (pudiendo estar también en contacto con otros habitantes infectados). En cuanto a los cuidadores, estos pasan cuatro horas en la zona común por lo que resulta más improbable que se contagien. Si bien ambos grupos pasan el resto de horas en casa y por tanto, potencialmente en contacto con alguna persona infectada, el riesgo es menor pues en una casa se puede producir contacto con tres personas infectadas como máximo mientras que, en la zona común o en la zona de trabajo se puede estar en contacto con decenas de personas infectadas lo que evidentemente aumenta el riesgo. Por tanto, en cuanto a la progresión de los infectados en una población podemos concluir que se debe fundamentalmente a dos factores.

1. El número de personas que conforma la población, a mayor número más probable es que se multiplique el número de contagios.

2. El rol o tareas que desempeñe cada habitante, el tiempo (número de horas) que un habitante está expuesto al contacto con otras personas infectadas es de vital importancia para determinar si se contagia o no.

Resulta interesante también comprobar cuál es el grupo (por rol) que presenta una mayor tasa de mortalidad. Las progresiones del número de fallecidos son las que pueden verse en la Figura 5.21. Como puede verse, el grupo de estudiantes presenta una cantidad de fallecidos comparable a la de los trabajadores, a pesar de ser el más numeroso. Esto es así porque, la regla que toma a un estudiante infectado asintomático y lo pasa a infectado en estado leve tiene un valor de la constante estocástica de 0.0005 mientras que, la regla que toma a un trabajador o un cuidador infectado asintomático y lo pasa a estado leve tiene una constante estocástica del 0.001. En otras palabras, con estos valores de tratamos de conseguir que el grupo de estudiantes, al ser un grupo de gente joven, tuviese una probabilidad menor de empeorar por contagiarse con el virus.

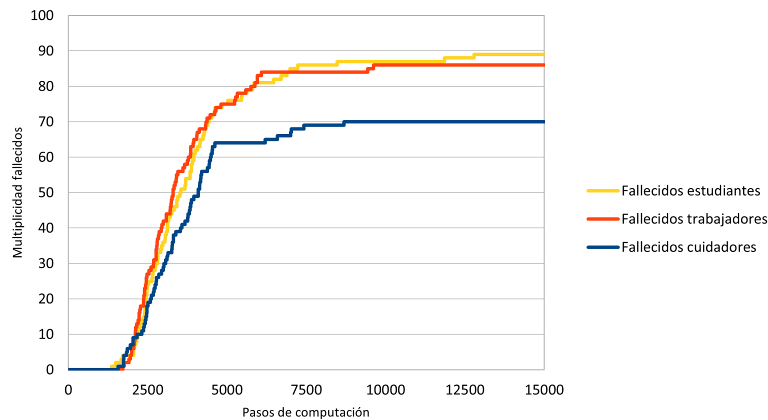


Figura 5.21: Progresión fallecidos por rol escenario COVID-19

Se observa además cierta diferencia en cuanto a la cantidad de fallecidos en el grupo de trabajadores con respecto a la del grupo de cuidadores. Al terminar la simulación, se alcanzó una cifra de 86 fallecidos trabajadores y 70 fallecidos cuidadores. Si bien ambos grupos tienen las mismas probabilidades de contagiarse, empeorar su estado y de fallecer, como vimos con las cifras de infectados, el grupo de los cuidadores tenía un menor número de infecciones que el de trabajadores. Esta diferencia entre el número de infectados explica porqué también tenemos al terminar un menor número de fallecidos.

Para terminar, vamos a examinar como avanza el desarrollo de inmunidad en función del rol que asuma cada habitante. Dicha progresión puede verse en la Figura 5.22 a continuación. La regla que genera la inmunidad a cada habitante tiene el mismo valor de la constante para todos ellos independientemente del rol que asuman, por tanto, tiene sentido que los grupos más numerosos desarrollen un mayor número de inmunes. Del proceso de inmunizado mencionar que, cualquier persona que esté infectada del virus puede desarrollar inmunidad independientemente del estado de gravedad. Es decir, cualquier habitante independientemente de si está asintomático, leve o grave puede curarse y en el proceso, desarrollar inmunidad al virus. Cuando un habitante se cura pueden ocurrir dos cosas, o bien que quede sano de nuevo pero no inmune al virus o bien que quede sano y además inmune al virus por lo que no podrá volver a contagiarse.

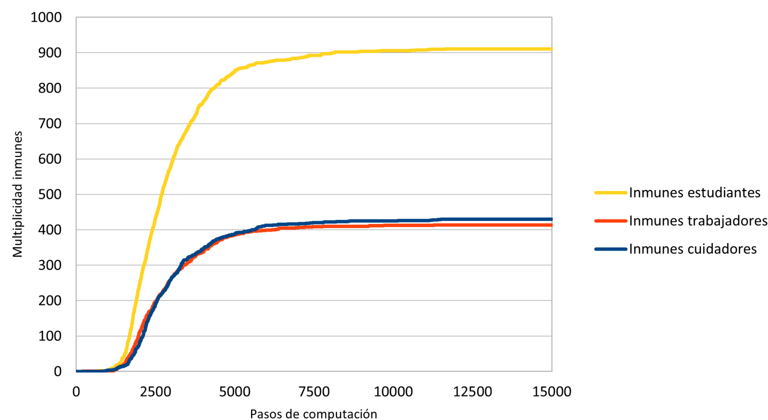


Figura 5.22: Progresión inmunizados por rol escenario COVID-19

Por todo esto, tiene sentido que el escenario quede estable únicamente cuándo un grupo lo suficientemente grande quede inmunizado al virus, entendiendo estable como un escenario en el que ya no se produce ningún contagio o al menos no se contagian

nuevos individuos. Cuando una persona se cura pero no desarrolla inmunidad al virus es susceptible a volver a contagiarse si entra en contacto con un nuevo infectado. En este contexto en el que existe una parte considerable de la población que no es inmune al virus es frecuente observar "repuntes" en la cantidad de infectados. Una prueba de esto es la gráfica que aparece en la Figura 5.23 dónde representamos la cantidad de trabajadores infectados frente a los pasos de computación.

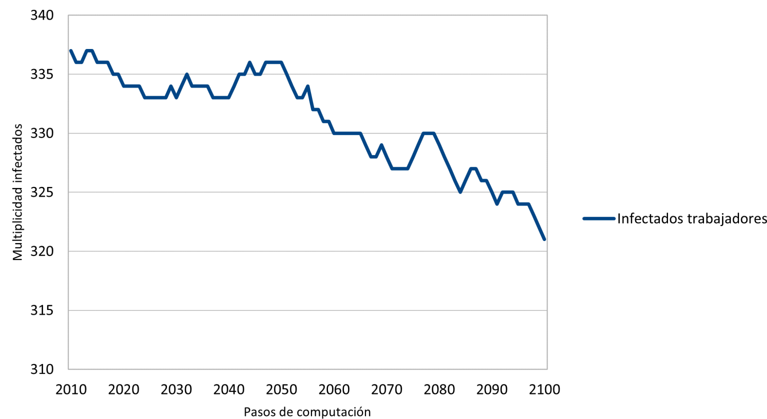


Figura 5.23: Repuntes en las infecciones de trabajadores escenario COVID-19

En esta figura, representamos 100 pasos de computación desde el paso 2010 hasta el 2110, lo que se correspondería a un intervalo desde el día 83 hora 18 hasta el día 87 hora 22. Durante este intervalo podemos ver descensos y ascensos de la cantidad de trabajadores infectados. Aunque, como vimos en la Figura 5.20 la tendencia es al descenso, es habitual encontrar ciertos ascensos en la cantidad de infectados. Como vimos al comienzo de esta sección 5.5, las personas sanas pueden contagiarse del virus y una persona que pase el virus puede curarse volviendo a quedar sana (y sin desarrollar inmunidad). Por tanto, para garantizar el cese de los contagios se debe conseguir un porcentaje cercano (aunque no necesariamente igual) al 100 % de la población inmunizada. Esto puede verse en la Figura 5.19 donde, las curvas tienden a quedar planas cuando la población sana se acerca a cero y la población inmune se acerca a la mayoría de habitantes que no han fallecido.

De este último hecho que hemos observado, se puede concluir que un protocolo que consiguiese inmunizar a la población sería una forma efectiva de frenar los contagios y poner fin a la pandemia. Un protocolo de vacunación de la población sería una forma de acometer esta problemática y podría resultar interesante incorporarlo a futuras simulaciones y recopilar información que sería potencialmente útil para estudiar el avance de la pandemia. Sin embargo, esto queda fuera del alcance de este proyecto y se deja para futuros trabajos en esta materia.

CAPÍTULO 6

Conclusiones

Para concluir este trabajo vamos a valorar el grado de cumplimiento de los objetivos que se propusieron en el capítulo de introducción. Por una parte, gracias a una lectura de un número considerable de artículos y capítulos de ciertos libros, se ha conseguido un grado de familiaridad elevado con los conceptos esenciales de la computación con membranas. Además, el estudio de ciertos modelos derivados de los sistemas P de transición, han ayudado a comprender el potencial de la computación con membranas y sus posibles aplicaciones en problemas del mundo real. Por otra parte, en cuanto a la adaptación de algoritmos de simulación estocásticos y probabilistas, podemos afirmar que hemos conseguido lograr parcialmente este objetivo. Como se desprende de lo desarrollado en este escrito, se han estudiado los sistemas P estocásticos y se han desarrollado un total de dos algoritmos que se probaron con diferentes escenarios. En lo que respecta a los sistemas P probabilistas, si bien se intentó desarrollar y adaptar el algoritmo DCBA (*Direct distribution based on Consistent Blocks Algorithm*) [11], finalmente, por falta de tiempo su implementación tuvo que interrumpirse y dejarse como un posible trabajo a futuro.

Por último, consideramos que sí se ha logrado con éxito estudiar los algoritmos implementados y lograr una simulación de un escenario de interés biológico como sería la expansión del virus COVID-19. En este sentido, el estudio de los cuatro primeros escenarios expuestos en el capítulo 5 nos permitió observar el comportamiento de los algoritmos, analizar los resultados obtenidos y determinar si existían diferencias significativas entre ambos. Todo este análisis permitió de esta forma determinar si eran aptos o no para la simulación de un escenario de interés biológico y, finalmente, aplicarlos a dicho escenario. Consideramos además que el estudio realizado en el escenario del COVID-19 nos ha permitido extraer una serie de conclusiones válidas, aplicables al mundo real y, en definitiva nos han llevado a considerar que los sistemas P estocásticos constituyen una herramienta adecuada y útil para conducir simulaciones de este tipo. Además, la capacidad de los sistemas P de abstraer ciertas realidades mediante el uso de objetos y membranas permite construir simulaciones ajustadas a las necesidades de cada momento, es decir, podríamos introducir en una simulación el uso de mascarillas, los confinamientos por zonas o la vacunación de individuos y observar cómo afectan estos parámetros a la expansión del virus. Sin embargo, esto queda fuera del alcance de este proyecto y se deja para futuros estudios en la materia.

Otro aspecto que consideramos importante es la adaptación del algoritmo 2 a la simulación del escenario COVID-19. Como vimos en el capítulo 5, el hecho de que este algoritmo ejecutase una única regla por bloque en cada paso de computación nos hacía perder cierta proporción entre bloques que sí mostraba el algoritmo 1. Argumentamos que esta diferencia hacía que ambos algoritmos se comportasen de forma diferente en presencia de más de un bloque de reglas y vimos también como su aplicación no resultaba posible en el último escenario. Pensamos sin embargo que es posible modificar el

algoritmo eliminando la restricción de ejecutar una regla por bloque y tratar de mantener esta proporción. Bajo estas condiciones, el algoritmo 2 podría resultar adecuado para la simulación del escenario COVID pero, esto de nuevo se reserva a futuros trabajos en la materia.

Bibliografía

[1] Besozzi, D., Cazzaniga, P., Pescini, D. and Mauri, G. (2008). Modelling metapopulations with stochastic membrane systems. Biosystems, 91(3), pp.499–514.

[2] Campos, M., Llorens, C., Sempere, J.M., Futami, R., Rodriguez, I., Carrasco, P., Capilla, R., Latorre, A., Coque, T.M., Moya, A. and Baquero, F. (2015). A membrane computing simulator of trans-hierarchical antibiotic resistance evolution dynamics in nested ecological compartments (ARES). Biology Direct, 10(1).

[3] Csuhaj-Varjú, E. and Vaszil, G. (n.d.). (2007) P Systems with String Objects and with Communication by Request. Membrane Computing, pp.228–239. Springer.

[4] Eleftherakis, G.K., Kefalas, P., Paun, G., Rozenberg, G. and Salomaa, A. (2007). 8th International Workshop on Membrane Computing. [online] dialnet.unirioja.es. Available at: https://dialnet.unirioja.es/servlet/libro?codigo=560172 [Accessed 13 Aug. 2021].

[5] Fontana, F., Bianco, L. and Manca, V. (2006). P Systems and the Modeling of Biochemical Oscillations. Membrane Computing, pp.199–208. Springer.

[6] García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J. and Riscos-Núñez, A. (2010). An Overview of P-Lingua 2.0. Membrane Computing, pp.264–288. Springer.

[7] Gexiang Zhang, Perez-JimenezM.J. and Gheorghe, M. (2017). Real-life Applications with Membrane Computing. Cham Springer International Publishing.

[8] J. Hopcroft and J. Ullman. (1979). Introduction to Automata Theory, Languages and Computation. Addison-Wesley Publishing Co.

[9] Leporati, A. (n.d.). (2007). (UREM) P Systems with a Quantum-Like Behavior: Background, Definition, and Computational Power. Membrane Computing, pp.32–53. Cham Springer International Publishing.

[10] Leporati, A., Zandron, C., Ferretti, C. and Mauri, G. (n.d.). (2007). Solving Numerical NP-Complete Problems with Spiking Neural P Systems. Membrane Computing, pp.336–352. Springer.

[11] Martínez-del-Amor, M.A., Pérez-Hurtado, I., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Romero-Jiménez, Á., Graciani, C., Riscos-Núñez, A., Colomer, M.A. and Pérez-Jiménez, M.J. (2013). DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution. Membrane Computing, pp.257–276. Springer

[12] Obtulowicz, A. Probabilistic P systems. Lecture Notes in Computer Science, 2597, 377–387, 2002. Springer.

[13] Obtułowicz, A. (2003). Probabilistic P Systems. Membrane Computing, pp.377–387. Springer.

[14] Păun, G. (2000). Computing with Membranes. Journal of Computer and System Sciences, 61(1), pp.108–143.

[15] Păun, G. (2001). P systems with active membranes: attacking NP-complete problems. J. Autom. Lang. Comb. 6, 1 (Jan. 2001), 75–90.

[16] Păun, A. and Păun, Gh. The power of communication: P systems with symport/antiport. New Generation Computing, 20, 3, (2002), 295–305.

[17] Pierluigi Frisco, Gheorghe, M. and Perez-JimenezM.J. (2014). Applications of Membrane Computing in Systems and Synthetic Biology. Cham Springer International Publishing.

[18] Romero-Campero, F.J. and Pérez-Jiménez, M.J. (2008). Modelling gene expression control using P systems: The Lac Operon, a case study. Biosystems, 91(3), pp.438–457.

[19] Romero-Campero, F.J. and Pérez-Jiménez, M.J. (2008). A Model of the Quorum Sensing System in Vibrio fischeri Using P Systems. Artificial Life, 14(1), pp.95–109.

[20] Rosen, K.H. (2019). Discrete mathematics and its applications. New York, Ny Mcgraw-Hill Education.

[21] Valencia Cabrera, L. (2014). Un entorno para la experimentacio n virtual con modelos computacionales basados en sistemas P. Tesis Doctoral. pp.25–26.Universidad de Sevilla.

[22] www.p-lingua.org. (n.d.). Main Page - The P-Lingua Website. [online] Available at: http://www.p-lingua.org/wiki/index.php/Main_Page [Accessed 29 Jul. 2021].