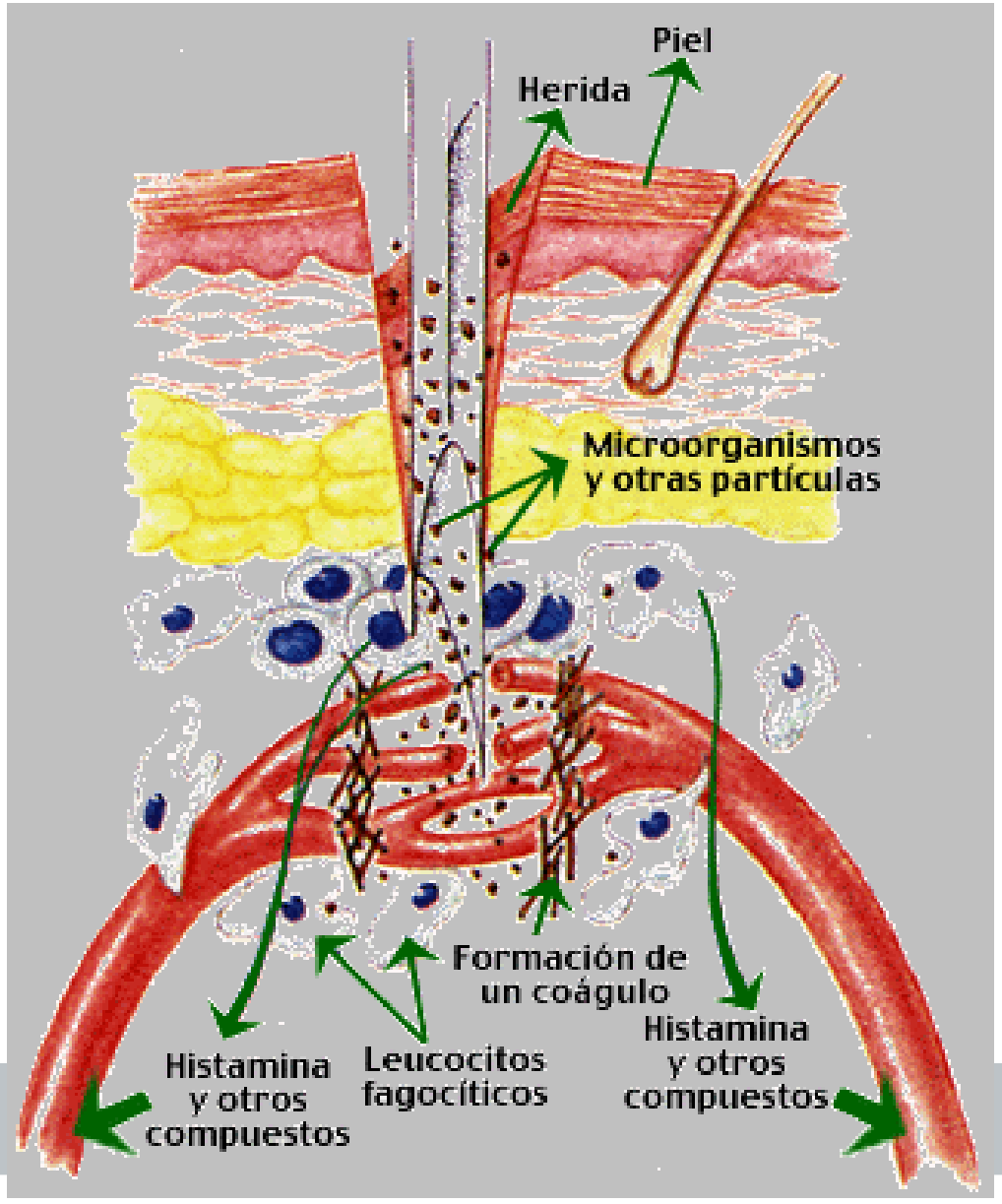
Computación Bioinspirada

Dr. Edward Hinojosa Cárdenas
ehinojosa@unsa.edu.pe

Sistemas Inmunológicos Natural

- El sistema inmunológico humano carga con la gran responsabilidad de detectar las infecciones o ataques de agentes externos (patógenos) a nuestro cuerpo, como bacterias y virus, y defendernos de los mismos, es decir, tiene la responsabilidad de mantener nuestro cuerpo saludable.

Sistemas Inmunológicos Natural



Sistemas Inmunológicos Natural

- Nuestro sistema inmunológico es un red compleja de órganos y células especializadas que trabajan en conjunto para combatir sustancias dañinas llamadas antígenos.

- Un antígeno es cualquier elemento que provoca una reacción inmunológica en nuestro cuerpo.

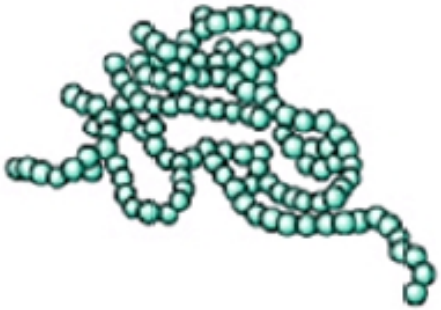
- El concepto de antígeno no debe confundirse con el concepto de patógeno.

Sistemas Inmunológicos Natural

- Un agente patógeno se refiere a un microorganismo vivo que invade nuestro cuerpo y libera uno o más antígenos, así un patógeno puede referirse a un virus, bacteria o microbio, mientras que un antígeno por lo general es una molécula.

- A continuación algunos ejemplos de patógenos:

Sistemas Inmunológicos Natural



Bacteria: Estreptococo

Virus: Herpes

Parásito: Esquistosoma

Hongo: Candida Albicans

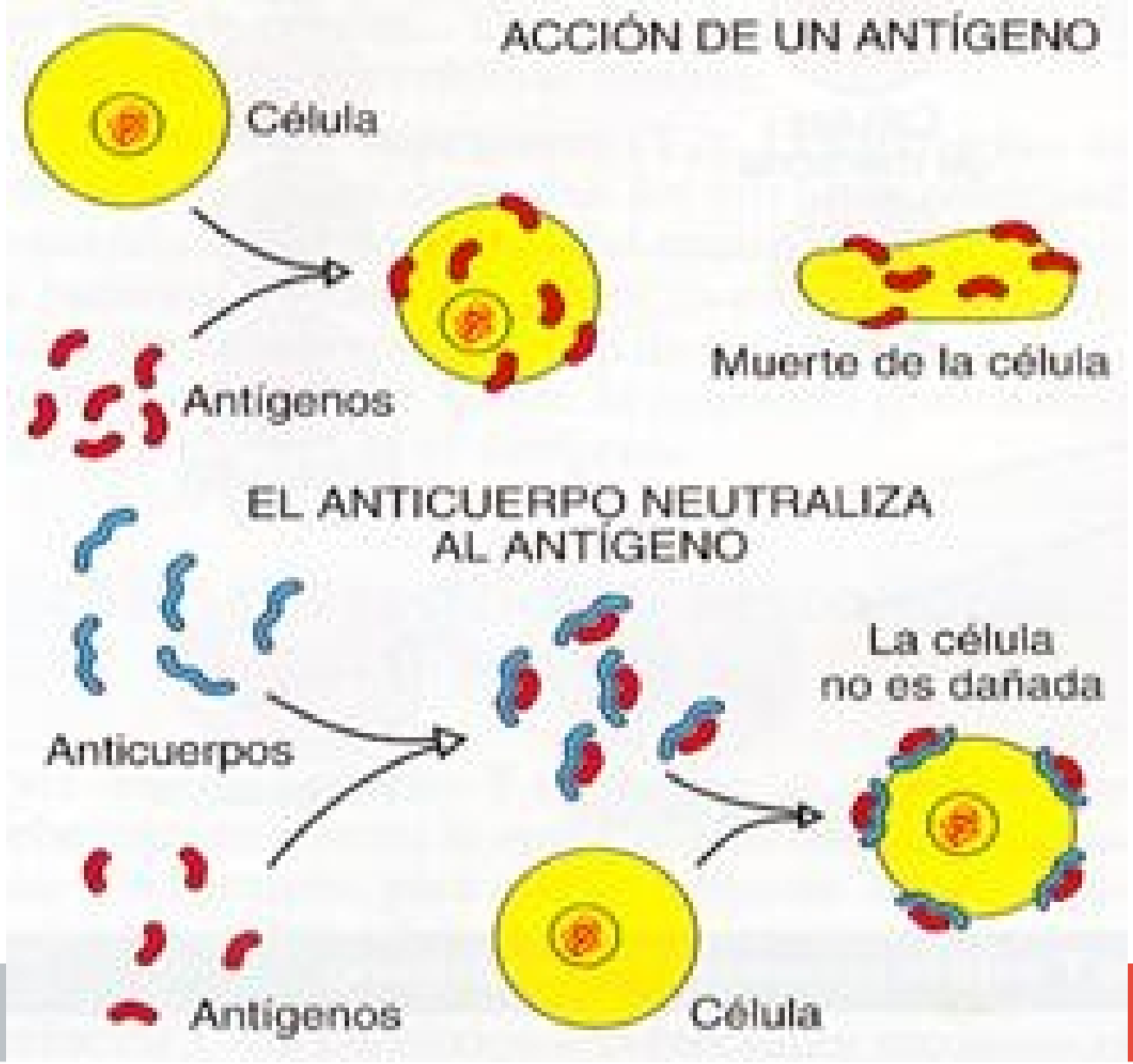
Sistemas Inmunológicos Natural

- La principal actividad que desarrolla el sistema inmunológico es la de eliminar los antígenos que son causantes de enfermedades.

- Para ello el sistema inmunológico libera en el cuerpo un grupo de células especiales que se encargan de reconocer y atacar a los antígenos, estas son las células conocidas como anticuerpos.

- En situaciones normales, nuestro sistema inmunológico diferencia entre lo que es “propio”, es decir, las células de nuestro cuerpo, de lo que no lo es.

Sistemas Inmunológicos Natural



Sistemas Inmunológicos Natural

- Los anticuerpos que libera el sistema inmunológico ante la presencia de antígenos determinan diferentes formas de respuestas inmunes, algunas tan específicas que los anticuerpos se han especializado en reconocer un tipo de bacteria de otra por medio de la experiencia.

- Esta forma de inmunidad específica se denomina inmunidad adquirida puesto que es activada únicamente ante la primera exposición de un patógeno en particular. Por otra parte, existen también respuestas naturales que fueron adquiridas antes del nacimiento, estas son conocidas como inmunidad innata.

Sistemas Inmunológicos Natural

- Los mecanismos de inmunidad innata se dividen en dos categorías. La primera de ellas está compuesta por las barreras de defensa pasiva (piel, mucosa, etc.).

- La segunda categoría de mecanismos de inmunidad innata son las respuestas activas que atacan de manera agresiva a cualquier agente extraño que entre en nuestro cuerpo.

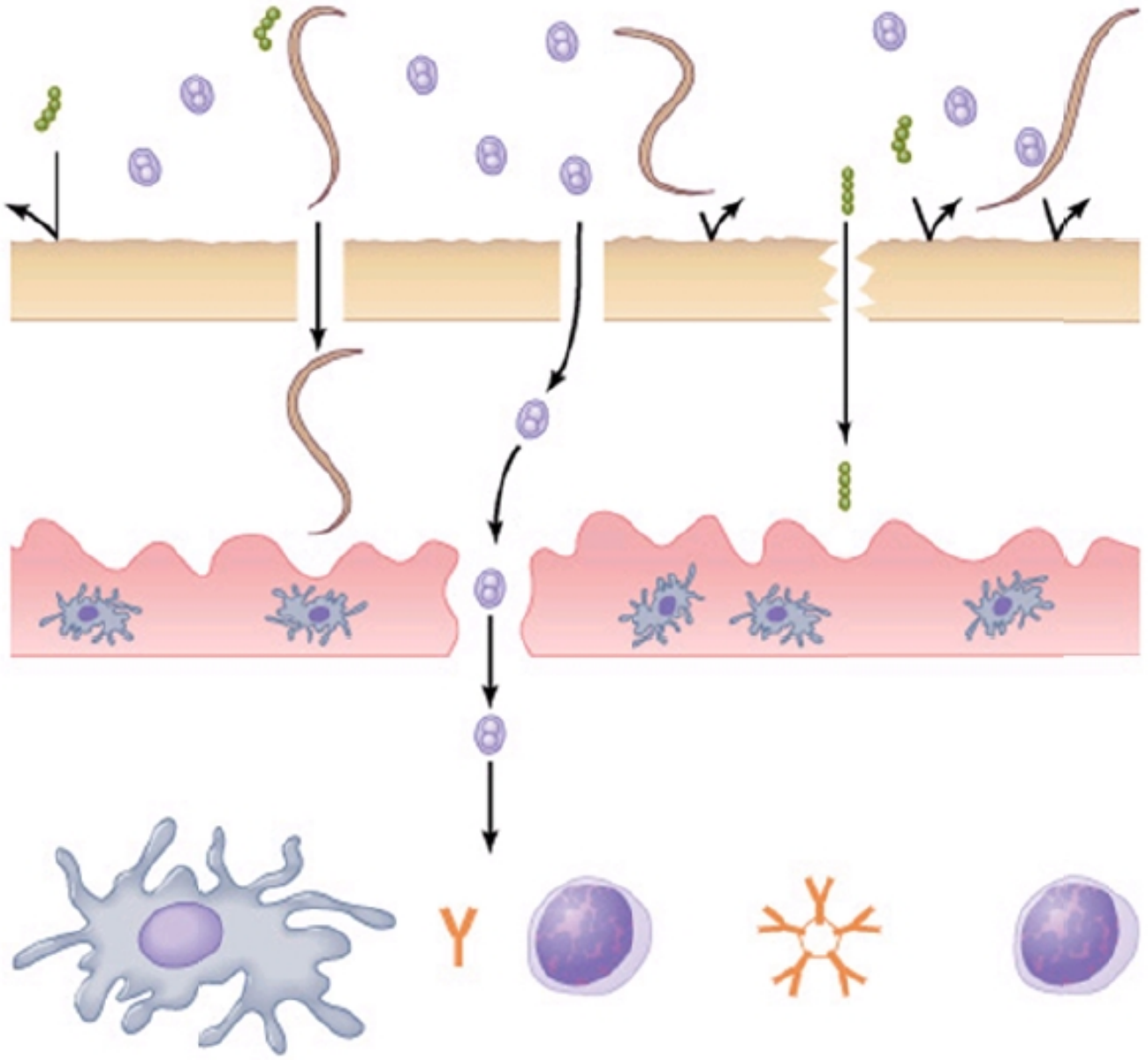
- De esta manera, el mecanismo de inmunidad innata puede visualizarse como un conjunto de barreras que impiden la entrada de intrusos al organismo.

Sistemas Inmunológicos Natural

Primera Línea:
Piel
Membranas Mucosas
Químicos

Segunda Línea:
Células Complementarias
Inflamación
Fiebre

Tercera Línea:
Linfocitos
Anticuerpos



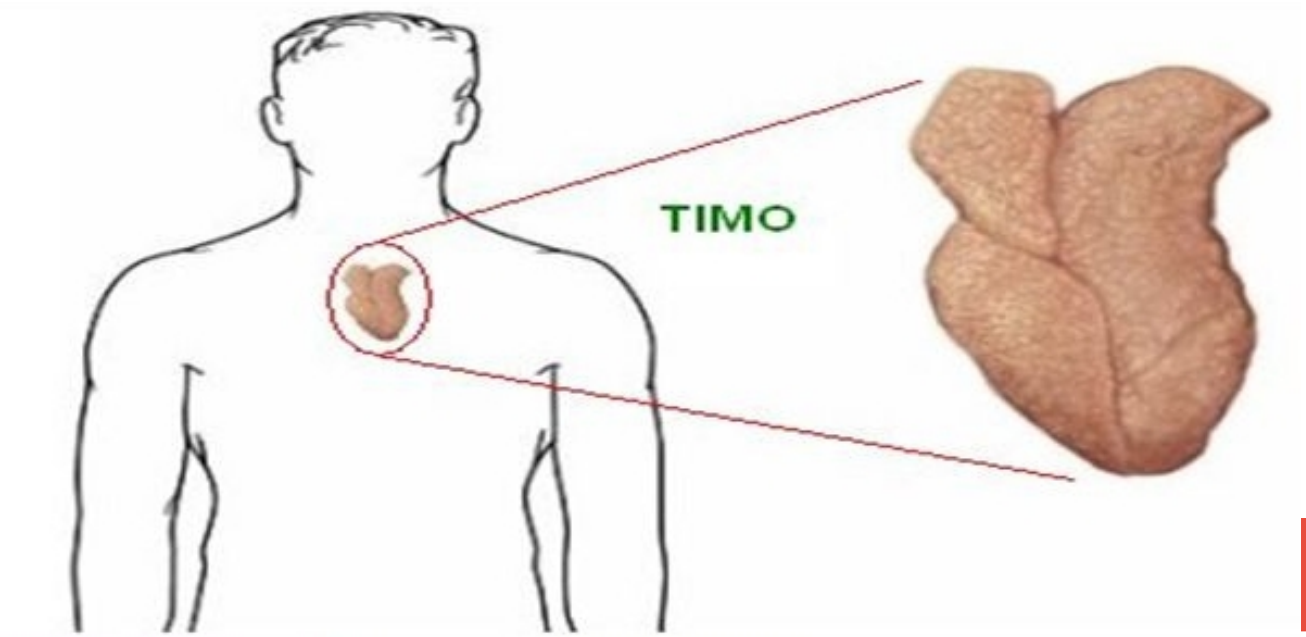
Sistemas Inmunológicos Natural

- Linfocitos: Células que permiten detectar y recordar invasores anteriores para contribuir a su destrucción.

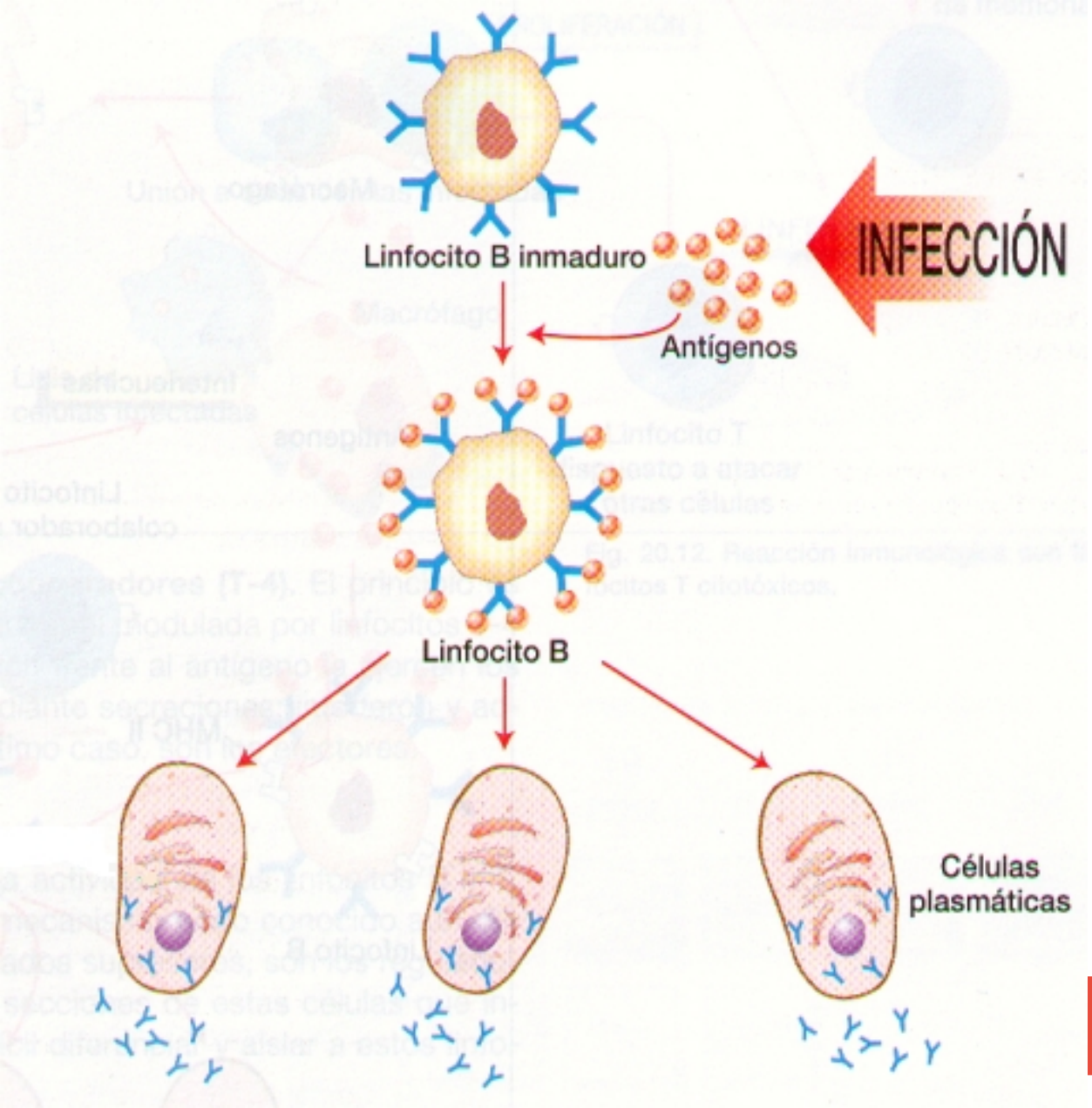
- Todas las células sanguíneas e inmunológicas, como en el caso de los linfocitos, se producen en la médula osea, que es un tejido blando y graso presente en las cavidades de los huesos.

Sistemas Inmunológicos Natural

- Una vez los linfocitos inician su formación, algunos continuarán su proceso de maduración y se volverán linfocitos B (llamados así porque maduran en la médula osea – Bone Marrow), otros terminarán su proceso de maduración en el timo y se convertirán en linfocitos T (llamados así porque maduran en el timo).



Sistemas Inmunológicos Natural



Sistemas Inmunológicos Natural

Linfocito T inmaduro

Linfocito T de ayuda

Linfocito T destructor

Sistemas Inmunológicos Artificiales

- Para que un sistema artificial pueda imitar la compleja labor del sistema inmune natural, éste debe estar dotado de un mecanismo de reconocimiento y de entrega de diferentes respuestas que dependen de la naturaleza del intruso, para que de esta forma se logre destruir o neutralizar los efectos negativos que este último pueda causar.

Sistemas Inmunológicos: Introducción

- Desde que los Sistemas Inmunológicos Artificiales fueron propuestos un gran número de modelos han surgido con el paso del tiempo, aplicados en áreas como optimización y clasificación, entre otras.

- Entre los modelos fundamentales y más estudiados se encuentran:
  - Algoritmo de Selección Clonal (CLONALG).
  - Algoritmo de Selección Negativa.

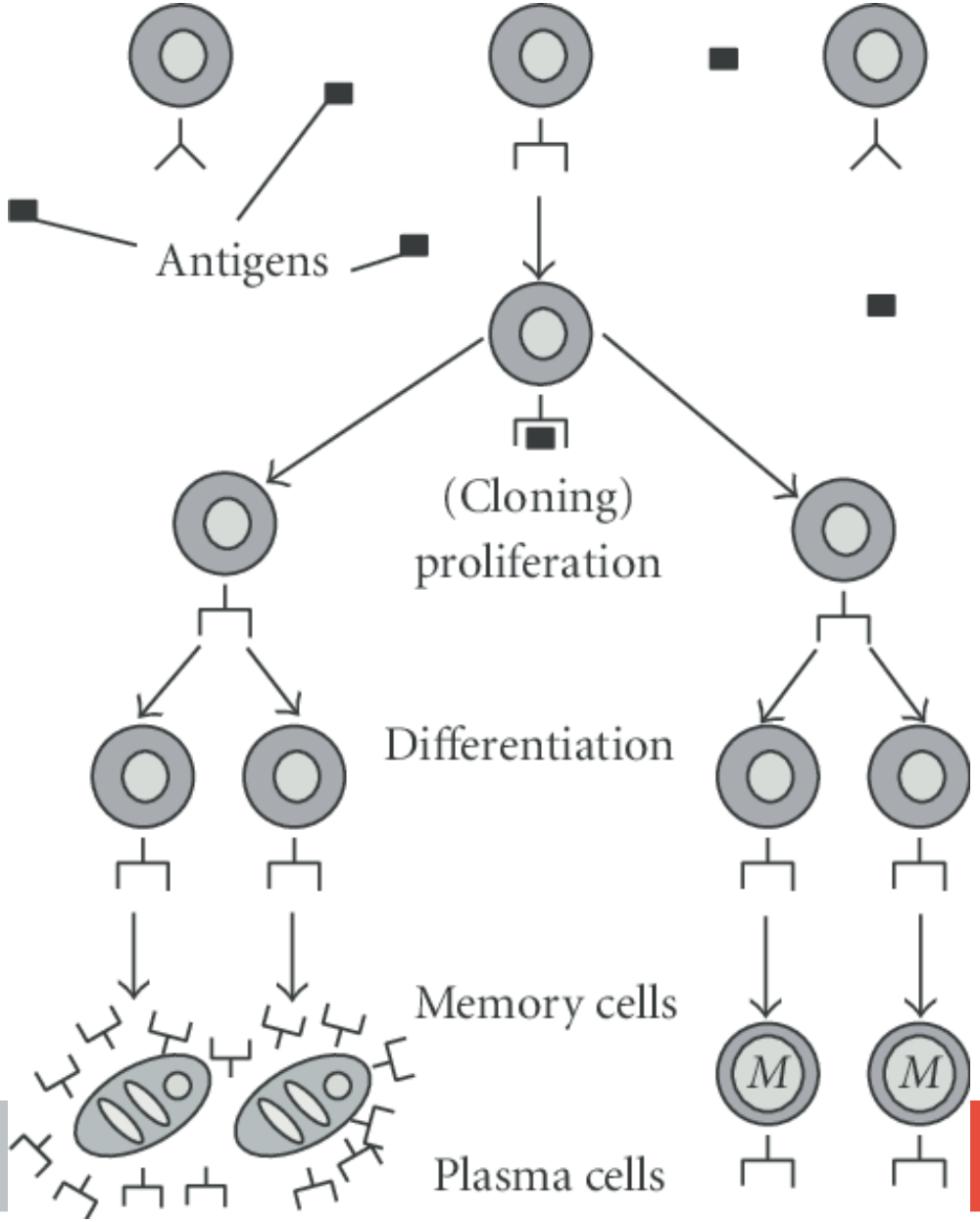
Algoritmo de Selección Clonal (CLONALG)

- El principio de la selección clonal describe las características básicas de la respuesta inmune ante un estímulo generado por un antígeno.

- Este principio establece la idea de que únicamente son proliferados aquellos linfocitos que reconocen a los antígenos, y no se presenta un proceso de competencia y selección entre linfocitos que reconocen antígenos y los que no.

Algoritmo de Selección Clonal (CLONALG)

- Una de las principales características de la teoría de la selección clonal es que los nuevos linfocitos son copias de sus padres (clones) y están sujetos a un mecanismo de mutación a determinadas tasas.



Algoritmo de Selección Clonal (CLONALG)

- En sistemas inmunológicos artificiales existe un algoritmo denominado CLONALG que está basado en el principio de selección clonal y maduración de la afinidad.

- Este algoritmo es similar a los algoritmos genéticos, pero sin cruzamiento.

Algoritmo de Selección Clonal (CLONALG)

- CLONal Selection ALGorithm (CLONALG).

- En la implementación del algoritmo CLONAG se asume un repertorio de anticuerpos (Ab) que pueden ser estimulados por un antígeno (o el valor de una función objetivo g(.) a ser optimizada) y los anticuerpos con una afinidad alta pueden ser seleccionados para generar poblaciones de clones.

Algoritmo de Selección Clonal (CLONALG)

- Durante el proceso de clonación, algunos anticuerpos pueden sufrir mutaciones somáticas proporcionales a su afinidad antigénica y los clones de afinidad alta serán seleccionados para formar el conjunto de memoria.

- Los anticuerpos de baja afinidad son reemplazados, simulando el proceso de edición del receptor.

Algoritmo de Selección Clonal (CLONALG)

```
Input: Population_size, Selection_size, Problem_size, RandomCells_num, Clone_rate, Mutation_rate
Output: Population
Population ← CreateRandomCells(Population_size, Problem_size)
While (¬StopCondition())
  For (p_i ∈ Population)
    Affinity(p_i)
  End
  Population_select ← Select(Population, Selection_size)
  Population_clones ← ∅
  For (p_i ∈ Population_select)
    Population_clones ← Clone(p_i, Clone_rate)
  End
  For (p_i ∈ Population_clones)
    Hypermutate(p_i, Mutation_rate)
    Affinity(p_i)
  End
  Population ← Select(Population, Population_clones, Population_size)
  Population_rand ← CreateRandomCells(RandomCells_num)
  Replace(Population, Population_rand)
End
Return (Population)
```

Pseudocode for CLONALG.

CLONALG: Ejemplo

Problema			
Minimizar	$f(x_1, x_2) = x_1^2 + x_2^2$	$LimIx_i$	$LimSx_i$
	$-5 \leq x_1 \leq 5$	-5	5
	$-5 \leq x_2 \leq 5$	-5	5
	$x_1, x_2 \in R$		

Algoritmo CLONALG: Ejemplo

- Considerar los siguientes parámetros:
  - StopCondition = 100 iteraciones
  - $Population_{size}$ = 4
  - $Selection_{size}$ = 4
  - $RandomCells_{num}$ = 2
  - $Clone_{rate}$ = 0.25
  - $Mutation_{factor}$ = -2.5
- Considerar codificación binaria (8 bits para cada número decimal)

CLONALG: Ejemplo

$$min + \frac{max - min}{(2{,}0^{8} \quad - 1{,}0)} * sum$$

$$sum = BinaryToDecimal$$

```
Población Inicial
{:bitstring=>"0000111111000100", :vector=>[-4.411764705882353, 2.686274509803922], :cost=>26.679738562091508}
{:bitstring=>"0010111101001011", :vector=>[-3.1568627450980395, -2.058823529411765], :cost=>14.204536716647446}
{:bitstring=>"0110010111001001", :vector=>[-1.0392156862745097, 2.88235294117647], :cost=>9.387927720107648}
{:bitstring=>"1101101111110101", :vector=>[3.5882352941176467, 4.607843137254902], :cost=>34.10765090349865}
```

CLONALG: Ejemplo

$$a = 1{,}0 - \frac{f}{max(f) - min(f)}$$

$$Cantidad\ Clones = Population_{zise} * Clone_{rate}$$

$$Mutation_{rate} = exp(Mutation_{factor} * a)$$

```
Iteracion 1
{:bitstring=>"1010011101110011", :vector=>[1.549019607843137, -0.4901960784313726], :cost=>2.6397539407920023, :affinity=>0.9215350155421467}
{:bitstring=>"1001011010101000", :vector=>[0.8823529411764701, 1.5882352941176467], :cost=>3.301038062283735, :affinity=>0.9018787712561712}
{:bitstring=>"1011100100110110", :vector=>[2.2549019607843137, -2.8823529411764706], :cost=>13.392541330257593, :affinity=>0.6019153410129822}
{:bitstring=>"0001111111110111", :vector=>[-3.784313725490196, 4.686274509803921], :cost=>36.282199154171465, :affinity=>-0.07846498445785333}
Clones (Número de Clones 1)
{:bitstring=>"0010011101110011", :vector=>[-3.4705882352941178, -0.4901960784313726], :cost=>12.285274894271435}
{:bitstring=>"1001011010101010", :vector=>[0.8823529411764701, 1.666666666666667], :cost=>3.556324490580546}
{:bitstring=>"1001000000111010", :vector=>[0.6470588235294112, -2.7254901960784315], :cost=>7.846981930026912}
{:bitstring=>"1110000000001000", :vector=>[3.784313725490197, -4.686274509803922], :cost=>36.28219915417148}
Población
{:bitstring=>"1010011101110011", :vector=>[1.549019607843137, -0.4901960784313726], :cost=>2.6397539407920023, :affinity=>0.9215350155421467}
{:bitstring=>"1001011010101000", :vector=>[0.8823529411764701, 1.5882352941176467], :cost=>3.301038062283735, :affinity=>0.9018787712561712}
{:bitstring=>"1001011010101010", :vector=>[0.8823529411764701, 1.666666666666667], :cost=>3.556324490580546}
{:bitstring=>"1001000000111010", :vector=>[0.6470588235294112, -2.7254901960784315], :cost=>7.846981930026912}
Inserción Aleatoria
{:bitstring=>"0010010010000100", :vector=>[-3.588235294117647, 0.17647058823529438], :cost=>12.906574394463668}
{:bitstring=>"0000100100110011", :vector=>[-4.647058823529412, -3.0], :cost=>30.595155709342563}
Población
{:bitstring=>"1010011101110011", :vector=>[1.549019607843137, -0.4901960784313726], :cost=>2.6397539407920023, :affinity=>0.9215350155421467}
{:bitstring=>"1001011010101000", :vector=>[0.8823529411764701, 1.5882352941176467], :cost=>3.301038062283735, :affinity=>0.9018787712561712}
{:bitstring=>"1001011010101010", :vector=>[0.8823529411764701, 1.666666666666667], :cost=>3.556324490580546}
{:bitstring=>"1001000000111010", :vector=>[0.6470588235294112, -2.7254901960784315], :cost=>7.846981930026912}
 > Generación 1, f=2.6397539407920023, s=[1.549019607843137, -0.4901960784313726]
```

CLONALG: Ejemplo

$$a = 1{,}0 - \frac{f}{max(f) - min(f)}$$

$$Cantidad\ Clones = Population_{zise} * Clone_{rate}$$

$$Mutation_{rate} = exp(Mutation_{factor} * a)$$

```
Iteracion 2
{:bitstring=>"1010011101110011", :vector=>[1.549019607843137, -0.4901960784313726], :cost=>2.6397539407920023, :affinity=>0.4930596574128767}
{:bitstring=>"1001011010101000", :vector=>[0.8823529411764701, 1.5882352941176467], :cost=>3.301038062283735, :affinity=>0.3660661547548733 6}
{:bitstring=>"1001011010101010", :vector=>[0.8823529411764701, 1.666666666666667], :cost=>3.556324490580546, :affinity=>0.3170407560543413 4}
{:bitstring=>"1001000000111010", :vector=>[0.6470588235294112, -2.7254901960784315], :cost=>7.846981930026912, :affinity=>-0.5069403425871233}
Clones (Número de Clones 1)
{:bitstring=>"0010011000010011", :vector=>[-3.5098039215686274, -4.254901960784314], :cost=>30.422914263744712}
{:bitstring=>"1001111111011110", :vector=>[1.2352941176470589, 3.7058823529411757], :cost=>15.259515570934251}
{:bitstring=>"0111110010111001", :vector=>[-0.13725490196078471, 2.2549019607843137], :cost=>5.103421760861207}
{:bitstring=>"0110111111000101", :vector=>[-0.6470588235294121, 2.7254901960784315], :cost=>7.846981930026914}
Población
{:bitstring=>"1010011101110011", :vector=>[1.549019607843137, -0.4901960784313726], :cost=>2.6397539407920023, :affinity=>0.4930596574128767}
{:bitstring=>"1001011010101000", :vector=>[0.8823529411764701, 1.5882352941176467], :cost=>3.301038062283735, :affinity=>0.3660661547548733 6}
{:bitstring=>"1001011010101010", :vector=>[0.8823529411764701, 1.666666666666667], :cost=>3.556324490580546, :affinity=>0.3170407560543413 4}
{:bitstring=>"0111110010111001", :vector=>[-0.13725490196078471, 2.2549019607843137], :cost=>5.103421760861207}
Inserción Aleatoria
{:bitstring=>"0111000110001001", :vector=>[-0.5686274509803919, 0.3725490196078427], :cost=>0.46212995001922275}
{:bitstring=>"1010000100101000", :vector=>[1.3137254901960782, -3.431372549019608], :cost=>13.500192233756248}
Población
{:bitstring=>"0111000110001001", :vector=>[-0.5686274509803919, 0.3725490196078427], :cost=>0.46212995001922275}
{:bitstring=>"1010011101110011", :vector=>[1.549019607843137, -0.4901960784313726], :cost=>2.6397539407920023, :affinity=>0.4930596574128767}
{:bitstring=>"1001011010101000", :vector=>[0.8823529411764701, 1.5882352941176467], :cost=>3.301038062283735, :affinity=>0.3660661547548733 6}
{:bitstring=>"1001011010101010", :vector=>[0.8823529411764701, 1.666666666666667], :cost=>3.556324490580546, :affinity=>0.3170407560543413 4}
 > Generación 2, f=0.46212995001922275, s=[-0.5686274509803919, 0.3725490196078427]
```

CLONALG: Ejemplo

$$a = 1{,}0 - \frac{f}{max(f) - min(f)}$$

$$Cantidad\ Clones = Population_{zise} * Clone_{rate}$$

$$Mutation_{rate} = exp(Mutation_{factor} * a)$$

```
Iteracion 100
{:bitstring=>"0111111110000010", :vector=>[-0.019607843137254832, 0.09803921568627416], :cost=>0.009996155324874977, :affinity=>0.7500000000000019}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
Clones (Número de Clones 1)
{:bitstring=>"0111101110000110", :vector=>[-0.17647058823529438, 0.2549019607843137], :cost=>0.09611687812379861}
{:bitstring=>"1000010001111100", :vector=>[0.17647058823529438, -0.13725490196078471], :cost=>0.049980776624375445}
{:bitstring=>"1000010001111100", :vector=>[0.17647058823529438, -0.13725490196078471], :cost=>0.049980776624375445}
{:bitstring=>"1000010001111100", :vector=>[0.17647058823529438, -0.13725490196078471], :cost=>0.049980776624375445}
Población
{:bitstring=>"0111111110000010", :vector=>[-0.019607843137254832, 0.09803921568627416], :cost=>0.009996155324874977, :affinity=>0.7500000000000019}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
Inserción Aleatoria
{:bitstring=>"1010001000111011", :vector=>[1.3529411764705879, -2.686274509803922], :cost=>9.04652056901192}
{:bitstring=>"1001011001111100", :vector=>[0.8823529411764701, -0.13725490196078471], :cost=>0.797385620915032}
Población
{:bitstring=>"0111111110000010", :vector=>[-0.019607843137254832, 0.09803921568627416], :cost=>0.009996155324874977, :affinity=>0.7500000000000019}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
{:bitstring=>"0111101110000011", :vector=>[-0.17647058823529438, 0.13725490196078383], :cost=>0.049980776624375195, :affinity=>-0.24999999999999822}
 > Generación 100, f=0.009996155324874977, s=[-0.019607843137254832, 0.09803921568627416]
Solucion Final: f=0.009996155324874977, s=[-0.019607843137254832, 0.09803921568627416]
```

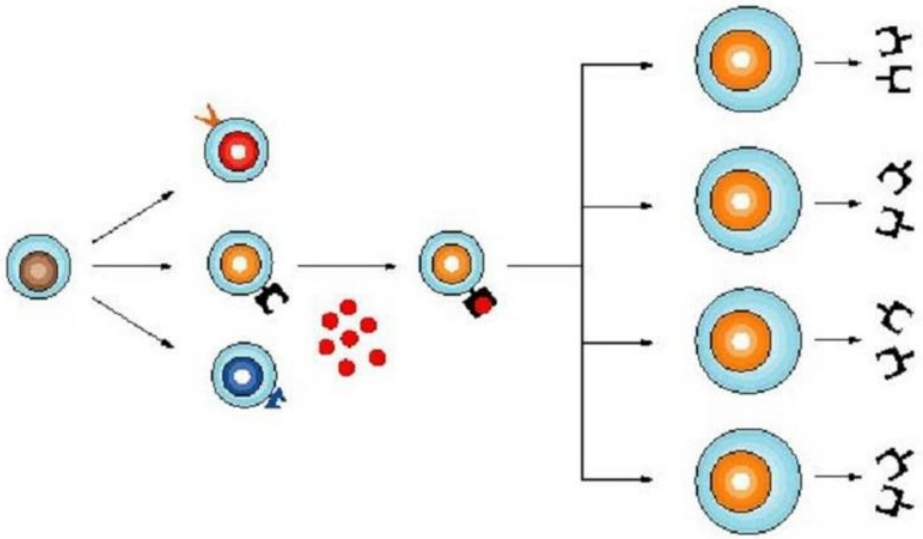
Algoritmo de Red Inmune

- Artificial Immune Network algorithm (aiNet) .

- El aiNet se inspira en la teoría de la Red Inmunológica del sistema inmunológico adquirido.

Algoritmo de Red Inmune

- La teoría de la selección clonal de la inmunidad adquirida explica el comportamiento adaptativo del sistema inmunológico, incluyendo la selección y proliferación continuas de células que seleccionan, para el material potencialmente dañino (y típicamente extraño) en el cuerpo.



Algoritmo de Red Inmune

- Una preocupación de la teoría de selección clonal es que presume que el repertorio de células reactivas permanece inactivo cuando no hay patógenos a los que responder.

- Jerne propuso una Teoría de la Red Inmune donde las células inmunes no están en reposo en ausencia de patógenos, sino que los anticuerpos y las células inmunes se reconocen y responden entre sí.

Algoritmo de Red Inmune

- La teoría de la Red Inmune propone que los anticuerpos poseen idiotopos (características de superficie) a los que se pueden unir los receptores de otros anticuerpos.

- Como resultado de las interacciones de los receptores, el repertorio se vuelve dinámico, donde los receptores se inhiben y excitan continuamente unos a otros en complejas redes reguladoras.

Algoritmo de Red Inmune

- La teoría sugiere que el proceso de selección clonal puede ser desencadenado por los idiotopos de otras células y moléculas inmunitarias además de las características superficiales del patógeno, y que el proceso de maduración se aplica tanto a los propios receptores como a los idiotopos que éstos exponen.

Optimization Artificial Immune Network algorithm (opt-aiNet)

```
Input: Population_size, ProblemSize, N_clones, N_random, AffinityThreshold
Output: S_best
Population ← InitializePopulation(Population_size, ProblemSize)
While (¬StopCondition())
  EvaluatePopulation(Population)
  S_best ← GetBestSolution(Population)
  Progeny ← ∅
  Cost_avg ← CalculateAveragePopulationCost(Population)
  While (CalculateAveragePopulationCost(Population) > Cost_avg)
    For (Cell_i ∈ Population)
      Clones ← CreateClones(Cell_i, N_clones)
      For (Clone_i ∈ Clones)
        Clone_i ← MutateRelativeToFitnessOfParent(Clone_i, Cell_i)
      End
      EvaluatePopulation(Clones)
      Progeny ← GetBestSolution(Clones)
    End
  End
  SupressLowAffinityCells(Progeny, AffinityThreshold)
  Progeny ← CreateRandomCells(N_random)
  Population ← Progeny
End
Return (S_best)
```

Pseudocode for opt-aiNet.

opt-aiNet

- El nivel de mutación de los clones es proporcional a la afinidad de la célula madre con la función de costo (mejor aptitud, menor mutación).

- Se considera una supresión basada en la similitud de las células para reducir la redundancia.

- Al final se adicionan células aleatorias en cada iteración para añadir una capacidad de reinicio aleatorio y evitar el estancamiento.

opt-aiNet

- El tamaño de la población es dinámico, y si continúa creciendo puede ser una indicación de un problema con muchas óptimas locales o que el umbral de afinidad necesita ser incrementado.

- La mutación se realiza con las siguientes fórmulas (beta – parámetro; f – costo normalizado):

$$c' = c + \alpha \times N(1,0)$$

$$\alpha = \frac{1}{\beta} \times exp(-f)$$

opt-aiNet

- El umbral de afinidad es específico del problema y de la representación, por ejemplo, a puede establecerse en un valor arbitrario como 0.1 en un dominio de función continua, o calcularse como un porcentaje del tamaño del espacio del problema.

- El número de celdas aleatorias insertadas puede ser el 40% del tamaño de la población.

- El número de clones creados para una célula puede ser pequeño, como 5.

opt-aiNet: Ejemplo

Minimizar	Problema	$Lim Ix_i$	$Lim Sx_i$
	$f(x_1, x_2) = x_1^2 + x_2^2$		
	$-5 \leq x_1 \leq 5$	-5	5
	$-5 \leq x_2 \leq 5$	-5	5
	$x_1, x_2 \in R$		

opt-aiNet: Ejemplo

```
Parámetros
Generaciones = 150
Population_Size = 5
Problem_size = 2
N_clones = 5
N_rand = 2
Affinity_threshold = 0.5; (l_max-l_min) * 0.05
Beta = 100
```

opt-aiNet: Ejemplo

```
Población Inicial
{:vector=>[3.676741527973938, 1.4643740007392445]}
{:vector=>[-0.3014582975370237, 4.654867106920243]}
{:vector=>[0.9359131076575817, 1.160521393167187]}
{:vector=>[-1.716350124113457, -0.13214292128245564]}
{:vector=>[3.6924756520689765, -4.6799419765842 47]}
```

opt-aiNet: Ejemplo

```
Iteración 1
Evaluar la población
{:vector=>[3.676741527973938, 1.4643740007392445], :cost=>15.66281947756919}
{:vector=>[-0.3014582975370237, 4.654867106920243], :cost=>21.758664888241952}
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805}
{:vector=>[-1.716350124113457, -0.13214292128245564], :cost=>2.963319500189341}
{:vector=>[3.6924756520689765, -4.679941976584247], :cost=>35.536233345317484}
Normalizar el costo
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805, :norm_cost=>0.9332779831034489}
{:vector=>[-1.716350124113457, -0.13214292128245564], :cost=>2.963319500189341, :norm_cost=>0.9110474618051387}
{:vector=>[3.676741527973938, 1.4643740007392445], :cost=>15.66281947756919, :norm_cost=>0.5298355281201814}
{:vector=>[-0.3014582975370237, 4.654867106920243], :cost=>21.758664888241952, :norm_cost=>0.3468512357790444}
{:vector=>[3.6924756520689765, -4.679941976584247], :cost=>35.536233345317484, :norm_cost=>-0.06672201689655122}
Obtener el mejor de la población
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805, :norm_cost=>0.9332779831034489}
Descendencia igual a null
Calcular el promedio del costo
15.62875609208039
```

opt-aiNet: Ejemplo

$$c' = c + \alpha \times N(1,0)$$

$$\alpha = \frac{1}{\beta} \times exp(-f)$$

```
Normalizar el costo
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805, :norm_cost=>0.9332779831034489}
{:vector=>[-1.716350124113457, -0.13214292128245564], :cost=>2.963319500189341, :norm_cost=>0.9110474618051387}
{:vector=>[3.676741527973938, 1.4643740007392445], :cost=>15.66281947756919, :norm_cost=>0.5298355281201814}
{:vector=>[-0.3014582975370237, 4.654867106920243], :cost=>21.758664888241952, :norm_cost=>0.3468512357790444}
{:vector=>[3.6924756520689765, -4.679941976584247], :cost=>35.536233345317484, :norm_cost=>-0.06672201689655122}
Obtener el mejor de la población
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805, :norm_cost=>0.9332779831034489}
Descendencia igual a null
Calcular el promedio del costo
15.62875609208039
Clones
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688}
{:vector=>[0.9300963044391801, 1.1643550087091612], :cost=>2.220801721837531}
{:vector=>[0.9394188997481563, 1.1635685396623956], :cost=>2.2363996156961163}
{:vector=>[0.9378336436084967, 1.1653352145934255], :cost=>2.237538105455494}
{:vector=>[0.9442823789134813, 1.1657324327770373], :cost=>2.2506013159547735}
```

```
Normalizar el costo
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805, :norm_cost=>0.9332779831034489}
{:vector=>[-1.716350124113457, -0.13214292128245564], :cost=>2.963319500189341, :norm_cost=>0.9110474618051387}
{:vector=>[3.676741527973938, 1.4643740007392445], :cost=>15.66281947756919, :norm_cost=>0.5298355281201814}
{:vector=>[-0.3014582975370237, 4.654867106920243], :cost=>21.758664888241952, :norm_cost=>0.3468512357790444}
{:vector=>[3.6924756520689765, -4.679941976584247], :cost=>35.536233345317484, :norm_cost=>-0.06672201689655122}
Obtener el mejor de la población
{:vector=>[0.9359131076575817, 1.160521393167187], :cost=>2.2227432490839805, :norm_cost=>0.9332779831034489}
Descendencia igual a null
Calcular el promedio del costo
15.62875609208039
Clones
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688}
{:vector=>[0.9300963044391801, 1.1643550087091612], :cost=>2.220801721837531}
{:vector=>[0.9394188997481563, 1.1635685396623956], :cost=>2.2363996156961163}
{:vector=>[0.9378336436084967, 1.1653352145934255], :cost=>2.237538105455494}
{:vector=>[0.9442823789134813, 1.1657324327770373], :cost=>2.2506013159547735}
Clones
{:vector=>[-1.7116448270468263, -0.12687242901769735], :cost=>2.945824627201O107}
{:vector=>[-1.7118537806070755, -0.13411027160119987], :cost=>2.948428931127685}
{:vector=>[-1.7130687435112615, -0.13819855968912348], :cost=>2.9537033618954003}
{:vector=>[-1.7145997922120166, -0.12274302047128391], :cost=>2.954918296527905}
{:vector=>[-1.7198169893651607, -0.14244113232098754], :cost=>2.97805995308593}
Clones
{:vector=>[3.6748362537915273, 1.4610816592931175], :cost=>15.639181107303276}
{:vector=>[3.6764598442210454, 1.4607965868168362], :cost=>15.650283654225552}
{:vector=>[3.673126158269759, 1.4704028797492579], :cost=>15.653940403340467}
{:vector=>[3.677955634821218, 1.461209773417547], :cost=>15.662491653644109}
{:vector=>[3.6786973863968555, 1.4672333940761586], :cost=>15.6855882933753}
Clones
{:vector=>[-0.3059973254964285, 4.645185307408854], :cost=>21.671380903378054}
{:vector=>[-0.29995709375241547, 4.64609715783346], :cost=>21.676193058120553}
{:vector=>[-0.3033956736855429, 4.648095604028771], :cost=>21.696841679002684}
{:vector=>[-0.291876546298765, 4.655646763606012], :cost=>21.760238705754432}
{:vector=>[-0.2922976880670901, 4.6613375797256005], :cost=>21.813505970611487}
Clones
{:vector=>[3.680363653834724, -4.680926087315013], :cost=>35.45614565937392}
{:vector=>[3.6849868368938554, -4.67873575264151], :cost=>35.469696231126896}
{:vector=>[3.687881083273484, -4.688395484523888], :cost=>35.581519103670395}
{:vector=>[3.6828667486648645, -4.694729032253173], :cost=>35.603988174702124}
{:vector=>[3.707343913470251, -4.696579250706939], :cost=>35.80225555091586}
Promedio Clones
15.586636746872705
```

opt-aiNet: Ejemplo

```
Clones Finales o Nueva Población
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688}
{:vector=>[-1.7116448270468263, -0.12687242901769735], :cost=>2.9458246272010107}
{:vector=>[3.6748362537915273, 1.4610816592931175], :cost=>15.639181107303276}
{:vector=>[-0.3059973254964285, 4.645185307408854], :cost=>21.671380903378054}
{:vector=>[3.680363653834724, -4.680926087315013], :cost=>35.45614565937392}
Nuevo Promedio
15.586636746872705
Población después del supresor de afinidad
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688}
{:vector=>[-1.7116448270468263, -0.12687242901769735], :cost=>2.9458246272010107}
{:vector=>[3.6748362537915273, 1.4610816592931175], :cost=>15.639181107303276}
{:vector=>[-0.3059973254964285, 4.645185307408854], :cost=>21.671380903378054}
{:vector=>[3.680363653834724, -4.680926087315013], :cost=>35.45614565937392}
```

opt-aiNet: Ejemplo

```
Insertar individuos aleatorios
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688}
{:vector=>[-1.7116448270468263, -0.1268724290176973 5], :cost=>2.9458246272010107}
{:vector=>[3.6748362537915273, 1.4610816592931175], :cost=>15.639181107303276}
{:vector=>[-0.3059973254964285, 4.645185307408854], :cost=>21.671380903378054}
{:vector=>[3.6803636538347 24, -4.680926087315013], :cost=>35.45614565937392}
{:vector=>[-4.224263384497192, 1.4743549001437888]}
{:vector=>[4.7931302267336235, -3.45240732799419]}
 > Generación 1, Tamaño_Población=7, Mejor_Costo=2.227432490839805
```

opt-aiNet: Ejemplo

```
Iteración 2
Evaluar la población
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688}
{:vector=>[-1.7116448270468263, -0.12687242901769735], :cost=>2.9458246272010107}
{:vector=>[3.6748362537915273, 1.4610816592931175], :cost=>15.639181107303276}
{:vector=>[-0.3059973254964285, 4.645185307408854], :cost=>21.671380903378054}
{:vector=>[3.680363653834724, -4.680926087315013], :cost=>35.45614565937392}
{:vector=>[-4.224263384497192, 1.4743549001437888], :cost=>20.018123513181674}
{:vector=>[4.7931302267336235, -3.45240732799419], :cost=>34.8932137288155}
Normalizar el costo
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688, :norm_cost=>0.9331843413473446}
{:vector=>[-1.7116448270468263, -0.12687242901769735], :cost=>2.9458246272010107, :norm_cost=>0.9113651023962385}
{:vector=>[3.6748362537915273, 1.4610816592931175], :cost=>15.639181107303276, :norm_cost=>0.5294434016020872}
{:vector=>[-4.224263384497192, 1.4743549001437888], :cost=>20.018123513181674, :norm_cost=>0.39768840567533326}
{:vector=>[-0.3059973254964285, 4.645185307408854], :cost=>21.671380903378054, :norm_cost=>0.3479446775050822}
{:vector=>[4.7931302267336235, -3.45240732799419], :cost=>34.8932137288155, :norm_cost=>-0.049877985730034435}
{:vector=>[3.680363653834724, -4.680926087315013], :cost=>35.45614565937392, :norm_cost=>-0.06681565865265537}
Obtener el mejor de la población
{:vector=>[0.9367347153165088, 1.1589562158374096], :cost=>2.2206514371072688, :norm_cost=>0.9331843413473446}
Descendencia igual a null
Calcular el promedio del costo
18.97778871090867
```

```
Clones
{:vector=>[0.9375834253373159, 1.1583753204938618], :cost=>2.2208960625965113}
{:vector=>[0.934904687028448, 1.1621923382553123], :cost=>2.2247378049271105}
{:vector=>[0.9340960219063865, 1.1634271507851448], :cost=>2.2260981133253765}
{:vector=>[0.9434710192712735, 1.1586625573912364], :cost=>2.232636486105176}
{:vector=>[0.9416205503820038, 1.1612997910382596], :cost=>2.2352664655672134}
Clones
{:vector=>[-1.7039562917325488, -0.1236302372982393], :cost=>2.918751479709358}
{:vector=>[-1.7095017942934558, -0.1304135402898287], :cost=>2.9394040761834717}
{:vector=>[-1.7103069887712332, -0.1271150308061893 3], :cost=>2.941308226896582}
{:vector=>[-1.71836613260929, -0.12928865183961818], :cost=>2.969497721193114}
{:vector=>[-1.7187345874375453, -0.1304971241250624 5], :cost=>2.9710780814590207}
Clones
{:vector=>[3.66248373065127, 1.4553744081120235], :cost=>15.531901745072668}
{:vector=>[3.6742192666750415, 1.4597850682734907], :cost=>15.63085966516032}
{:vector=>[3.676425387057595, 1.4567955533308357], :cost=>15.638356910806081}
{:vector=>[3.6739361553174925, 1.4654493859668567], :cost=>15.645348776179716}
{:vector=>[3.6750127244057453, 1.4669388857952905], :cost=>15.657628219202469}
Clones
{:vector=>[-4.214242810504136, 1.4672440472008073], :cost=>19.912647559932005}
{:vector=>[-4.215298150923597, 1.4666716996881808], :cost=>19.919864375846114}
{:vector=>[-4.21640420961572, 1.4666124605568547], :cost=>19.9290165683258}
{:vector=>[-4.219321233876329, 1.4737807497493984], :cost=>19.974701372971566}
{:vector=>[-4.225573627302277, 1.4829746588338624], :cost=>20.05468631849593}
Clones
{:vector=>[-0.3058129930638175, 4.64122482589608], :cost=>21.634489471240748}
{:vector=>[-0.307211203463168, 4.641871139746212], :cost=>21.641346401542087}
{:vector=>[-0.30076389623106864, 4.643952078090443], :cost=>21.656749824876638}
{:vector=>[-0.30571174242752525, 4.645339599299675], :cost=>21.672639662279735}
{:vector=>[-0.3068718957397429, 4.653333503260113], :cost=>21.74768305295794}
Clones
{:vector=>[4.787484281126433, -3.443265269933638], :cost=>34.77608146116385}
{:vector=>[4.796041554816683, -3.447142136788169], :cost=>34.88480350674893}
{:vector=>[4.7893145380283215, -3.458472508745144], :cost=>34.89856583791536}
{:vector=>[4.793875240935663, -3.454681751246501], :cost=>34.91606582805156}
{:vector=>[4.799100270825019, -3.4484487761683114], :cost=>34.9231623712895}
Clones
{:vector=>[3.6801908864762565, -4.673741693187165], :cost=>35.38766637553893}
{:vector=>[3.674759358953278, -4.682515602272556], :cost=>35.42980871174062}
{:vector=>[3.6750205919150725, -4.684218566365238], :cost=>35.44767992848061}
{:vector=>[3.6835706108212984, -4.681713094198618], :cost=>35.48712994129719}
{:vector=>[3.6861611107891794, -4.691465589083347], :cost=>35.59763310824768}
Promedio Clones
18.91177630789344
```

opt-aiNet: Ejemplo

```
Clones Finales o Nueva Población
{:vector=>[0.9375834253373159, 1.1583753204938618], :cost=>2.2208960625965113}
{:vector=>[-1.7039562917325488, -0.1236302372982393], :cost=>2.918751479709358}
{:vector=>[3.66248373065127, 1.4553744081120235], :cost=>15.531901745072668}
{:vector=>[-4.214242810504136, 1.4672440472008073], :cost=>19.912647559932005}
{:vector=>[-0.3058129930638175, 4.64122482589608], :cost=>21.634489471240748}
{:vector=>[4.787484281126433, -3.443265269933638], :cost=>34.77608146116385}
{:vector=>[3.6801908864762565, -4.673741693187165], :cost=>35.38766637553893}
Nuevo Promedio
18.91177630789344
Población después del supresor de afinidad
{:vector=>[0.9375834253373159, 1.1583753204938618], :cost=>2.2208960625965113}
{:vector=>[-1.7039562917325488, -0.1236302372982393], :cost=>2.918751479709358}
{:vector=>[3.66248373065127, 1.4553744081120235], :cost=>15.531901745072668}
{:vector=>[-4.214242810504136, 1.4672440472008073], :cost=>19.912647559932005}
{:vector=>[-0.3058129930638175, 4.64122482589608], :cost=>21.634489471240748}
{:vector=>[4.787484281126433, -3.443265269933638], :cost=>34.77608146116385}
{:vector=>[3.6801908864762565, -4.673741693187165], :cost=>35.38766637553893}
Insertar individuos aleatorios
{:vector=>[0.9375834253373159, 1.1583753204938618], :cost=>2.2208960625965113}
{:vector=>[-1.7039562917325488, -0.1236302372982393], :cost=>2.918751479709358}
{:vector=>[3.66248373065127, 1.4553744081120235], :cost=>15.531901745072668}
{:vector=>[-4.214242810504136, 1.4672440472008073], :cost=>19.912647559932005}
{:vector=>[-0.3058129930638175, 4.64122482589608], :cost=>21.634489471240748}
{:vector=>[4.787484281126433, -3.443265269933638], :cost=>34.77608146116385}
{:vector=>[3.6801908864762565, -4.673741693187165], :cost=>35.38766637553893}
{:vector=>[-2.4458085034315804, 4.791604369368722]}
{:vector=>[-3.484718011191293, 2.080029233471957]}
 > Generación 2, Tamaño_Población=9, Mejor_Costo=2.2206514371072688
```

opt-aiNet: Ejemplo

```
Clones Finales o Nueva Población
{:vector=>[-0.3856851410484299, 0.6370528438256005], :cost=>0.5545893538518321}
{:vector=>[2.1855223436706277, -0.8335488625204749], :cost=>5.471311620892731}
{:vector=>[0.9091410366372729, 2.1826735548879332], :cost=>5.590601271705022}
{:vector=>[1.7591404520541345, 1.8283266253238222], :cost=>6.4373533789212205}
{:vector=>[-1.554588620327042, 2.354220725139784], :cost=>7.959101001128026}
{:vector=>[4.0133547065227715, 0.33176457385838026], :cost=>16.217083732835913}
{:vector=>[1.4855661988371962, 3.9385350019178293], :cost=>17.71896489245947}
{:vector=>[2.7633479625563373, -3.5173485020333004], :cost=>20.007832446920162}
{:vector=>[0.6153515618786563, -4.525801183574579], :cost=>20.861533897951563}
{:vector=>[-4.525620062013486, -0.6449273643939721], :cost=>20.897168251043105}
{:vector=>[-1.1557168849443458, -4.603096947433673], :cost=>22.524183025618658}
{:vector=>[-1.6045420740626524, -4.636976087262988], :cost=>24.076102501286048}
Nuevo Promedio
14.026318781217812
Población después del supresor de afinidad
{:vector=>[-0.3856851410484299, 0.6370528438256005], :cost=>0.5545893538518321}
{:vector=>[2.1855223436706277, -0.8335488625204749], :cost=>5.471311620892731}
{:vector=>[0.9091410366372729, 2.1826735548879332], :cost=>5.590601271705022}
{:vector=>[1.7591404520541345, 1.8283266253238222], :cost=>6.4373533789212205}
{:vector=>[-1.554588620327042, 2.354220725139784], :cost=>7.959101001128026}
{:vector=>[4.0133547065227715, 0.33176457385838026], :cost=>16.217083732835913}
{:vector=>[1.4855661988371962, 3.9385350019178293], :cost=>17.71896489245947}
{:vector=>[2.7633479625563373, -3.5173485020333004], :cost=>20.007832446920162}
{:vector=>[0.6153515618786563, -4.525801183574579], :cost=>20.861533897951563}
{:vector=>[-4.525620062013486, -0.6449273643939721], :cost=>20.897168251043105}
{:vector=>[-1.1557168849443458, -4.603096947433673], :cost=>22.524183025618658}
```

opt-aiNet: Ejemplo

```
{:vector=>[4.413418902761675, -4.156183378569977], :cost=>36.75212668755542}
{:vector=>[-4.231594181762175, 4.573075217745805], :cost=>38.81940626628433}
{:vector=>[1.4323510754411704, 2.2339434199224915]}
{:vector=>[-1.9318068893477056, 0.0003739373584537 02]}
 > Generación 150, Tamaño_Población=106, Mejor_Costo=9.754245959210189e-08
Solución: Mejor_Costo=9.754245959210189e-08, s=[0.0001585342157339125, -0.0002690898724129493]
```

Algoritmo de Selección Negativa

- Este algoritmo utiliza las propiedades del sistema inmunológico para distinguir las células extrañas (no propias) de las células que pertenecen al cuerpo.

- Esta característica se puede utilizar para distinguir patrones normales de patrones anormales en un sistema, proporcionando así un mecanismo de detección de fallos.

Algoritmo de Selección Negativa

- Esta característica puede ser ventajosa cuando el el comportamiento normal de un sistema se define por un conjunto de patrones complejos, donde es muy difícil obtener relaciones.

- En este caso, puede ser más fácil observar patrones anormales que los normales.

Algoritmo de Selección Negativa

- También puede ser ventajoso en sistemas donde el número de posibles patrones anormales es mucho mayor que el número de patrones normales.

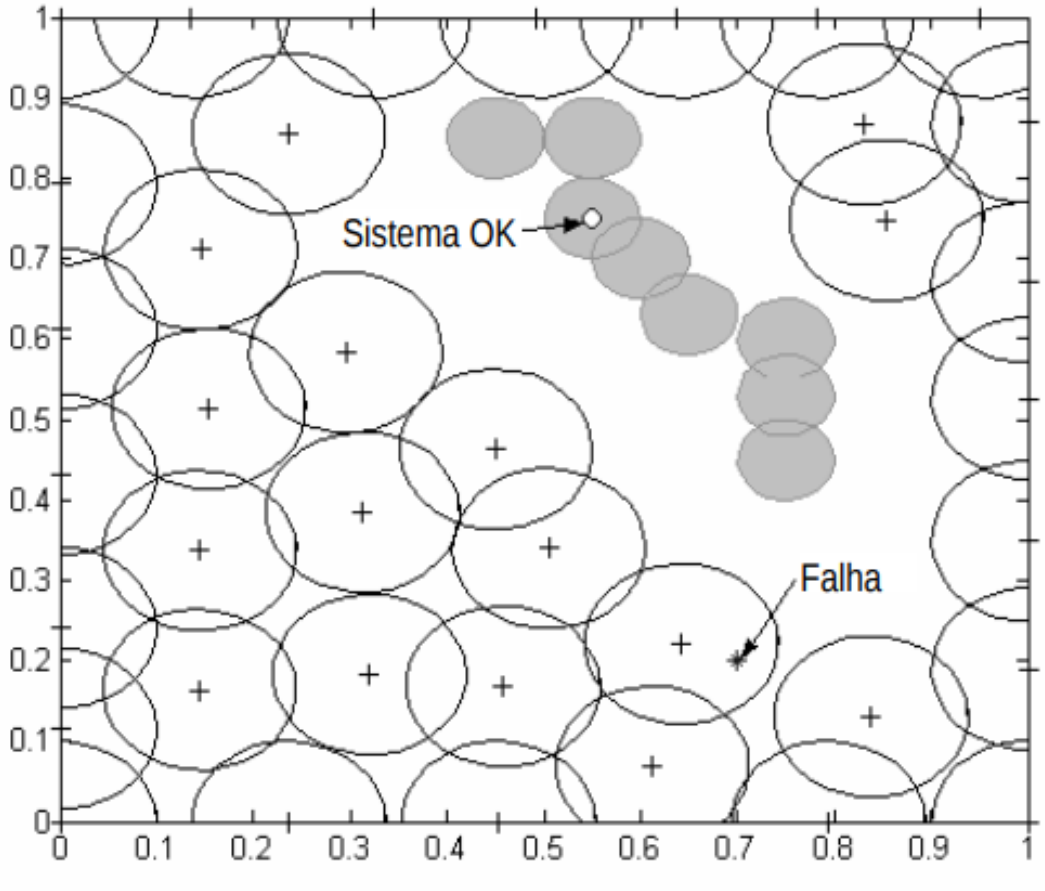
- Dado que la formación de un sistema de detección de fallos con un gran número de situaciones de fallo resulta poco práctica, es aconsejable detectar cualquier comportamiento anormal y, a continuación, tratar de identificar su causa.

Algoritmo de Selección Negativa

- El Algoritmo de Selección Negativa (NSA) se inspira en el mecanismo utilizado por el sistema inmunológico para entrenar a las células T a reconocer antígenos (no propios) y evitar que vuelvan a reconocer las células que pertenecen al cuerpo (propias).

- La idea básica es generar un conjunto de detectores (binarios) creando primero candidatos y luego desechando los que corresponden a la noción de uno mismo. Estos detectores se pueden utilizar posteriormente para detectar anomalías.

Algoritmo de Selección Negativa



Cobertura dos Detectores

Algoritmo de Selección Negativa

- El algoritmo de selección negativa fue diseñado para la detección de cambios, detección de novedad, detección de intrusos y reconocimiento de patrones similares y dominios de problemas de clasificación de dos clases.

- Los algoritmos tradicionales de selección negativa utilizaban representaciones binarias y reglas de correspondencia binaria como la distancia de euclideana y los bits contiguos.

Algoritmo de Selección Negativa

- Se debe seleccionar una representación de datos que sea la más adecuada para un dominio de problema dado y, a su vez, se selecciona una regla de correspondencia o se adapta a la representación de datos.

- Los detectores pueden prepararse sin ningún conocimiento previo del dominio del problema, salvo el conjunto de datos conocido (normal o propio).

NSA: Algoritmo

```
Input: SelfData
Output: Repertoire
Repertoire ← ∅
While (¬StopCondition())
  Detectors ← GenerateRandomDetectors()
  For (Detector_i ∈ Repertoire)
    If (¬Matches(Detector_i, SelfData))
      Repertoire ← Detector_i
    End
  End
End
Return (Repertoire)
```

Pseudocode for detector generation.

```
Input: InputSamples, Repertoire
For (Input_i ∈ InputSamples)
  Inputi_class ← "non-self"
  For (Detector_i ∈ Repertoire)
    If (Matches(Input_i, Detector_i))
      Inputi_class ← "self"
      Break
    End
  End
End
```

Pseudocode for detector application.

NSA: Ejemplo

- Como ejemplo se propone un problema sencillo de clasificación de dos clases en el que las muestras se extraen de un dominio bidimensional, donde:

$$x_i \in [0,1]$$

NSA: Ejemplo

- Los ejemplos entre:

$$1.0 > x_i > 0.5$$

- Son considerados como propios.

- El resto son considerados como no propios.

NSA: Ejemplo

- Parámetros:
  - Patrones de Identidad: 20
  - Cantidad de Detectores: 60
  - Distancia mínima: 0.05
  - Ejemplos a clasificar: 20

NSA: Ejemplo

```
Definimos los 20 patrones de identidad
{:vector=>[0.84677065460628l2, 0.9625993900798682]}
{:vector=>[0.5230674146432088, 0.6559412394216756]}
{:vector=>[0.7465878345643974, 0.5465816446128691]}
{:vector=>[0.7095779768637319, 0.93838567671052761]}
{:vector=>[0.8287437341219225, 0.887244372418843]}
{:vector=>[0.9455261460861997, 0.6627672910484501]}
{:vector=>[0.5632037883651396, 0.5514091633137594]}
{:vector=>[0.6254872906044032, 0.8838257246733543]}
{:vector=>[0.6922927366231607, 0.8605199402126877]}
{:vector=>[0.545274411589974l, 0.6325166500209832]}
{:vector=>[0.7745729441721597, 0.6557246105033215]}
{:vector=>[0.9774725511680337, 0.9101999885889912]}
{:vector=>[0.6740311462122012, 0.7373908265682917]}
{:vector=>[0.963448226483278, 0.5494016193898464]}
{:vector=>[0.85959684300723, 0.6745859508108333]}
{:vector=>[0.5454560558269591, 0.5210279610590944]}
{:vector=>[0.8324774589438146, 0.982774911996315]}
{:vector=>[0.6601129618002612, 0.7537400399196195]}
{:vector=>[0.7515969027282171, 0.5262492280267168]}
{:vector=>[0.7578382669696702, 0.5924083958037649]}
```

NSA: Ejemplo

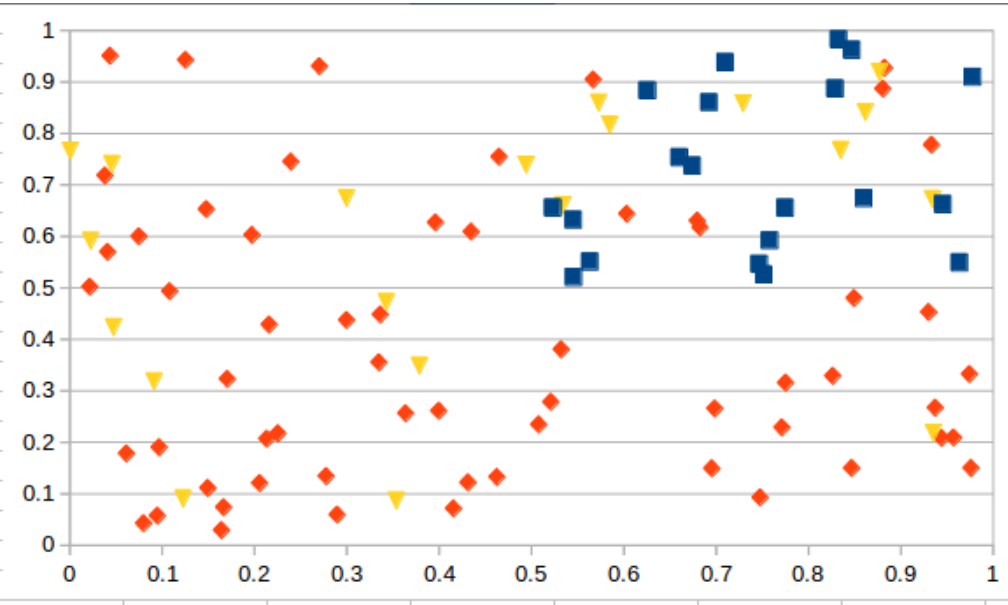
```
Definimos los 60 detectores.
{:vector=>[0.6032241920712659, 0.6439385366807321]}
{:vector=>[0.3960579414744744, 0.6271517523473934]}
{:vector=>[0.9332820066253557, 0.7778238921063105]}
{:vector=>[0.07998472505688314, 0.04281001609975188]}
{:vector=>[0.021800040730240666, 0.5023030577137739]}
{:vector=>[0.0749266871464559, 0.6000017417045913]}
{:vector=>[0.7712641297129439, 0.22876426954249718]}
{:vector=>[0.22539562483884368, 0.2174297313423812]}
{:vector=>[0.04363892417258708, 0.950822495686869]}
{:vector=>[0.09496628636761284, 0.057384660604914 54]}
{:vector=>[0.28963214757655464, 0.05941014821386603]}
{:vector=>[0.1478952030826156, 0.6528575230818703]}
{:vector=>[0.67961631738128, 0.6304419592580105]}
{:vector=>[0.9743398713884958, 0.33252474371881835]}
{:vector=>[0.43478637307080237, 0.6092017895543229]}
{:vector=>[0.5209602213678115, 0.27840263186250 17]}
{:vector=>[0.4626149953503086, 0.1325889271184063]}
{:vector=>[0.23944980762149826, 0.745212364168036]}
{:vector=>[0.09689895098388823, 0.1901425152795433]}
{:vector=>[0.507818544042776, 0.23452770620999097]}
{:vector=>[0.12541130947230217, 0.9428059662039654]}
{:vector=>[0.39966595454857246, 0.26113758150635924]}
{:vector=>[0.17059082880065257, 0.322996333785869]}
{:vector=>[0.21338342224948714, 0.20688934494028155]}
{:vector=>[0.0381463619146416, 0.7180984679280238]}
{:vector=>[0.9571495076682731, 0.20923707399457103]}
{:vector=>[0.27772634339879954, 0.13433020995338063]}
{:vector=>[0.6825222986033456, 0.617819006082322]}
{:vector=>[0.04104138490075704, 0.5699675819586374]}
{:vector=>[0.9760378457378417, 0.1500427296064002]}
{:vector=>[0.16670442726559853, 0.07380360764278315]}
{:vector=>[0.3362445299550625, 0.448135628044306 73]}
{:vector=>[0.2704120380263525, 0.9306404171194034]}
{:vector=>[0.20569778591654997, 0.1208921289691125]}
{:vector=>[0.061445992379168146, 0.178213325509861 13]}
```

NSA: Ejemplo

```
Generamos 20 nuevo ejemplos
{:vector=>[0.045608737397678345, 0.7404296960715924]}: predicho=N, esperado=N
{:vector=>[0.7292523233154654, 0.8587032177025028]}: predicho=S, esperado=S
{:vector=>[0.5848728648331275, 0.8169421022545066]}: predicho=S, esperado=N
{:vector=>[0.022722722239640938, 0.5913455926516042]}: predicho=N, esperado=N
{:vector=>[0.49426058584309485, 0.7393788344632162]}: predicho=N, esperado=N
{:vector=>[0.8349491752033195, 0.7674795870752397]}: predicho=S, esperado=N
{:vector=>[0.3787082784299086, 0.348724090347173]}: predicho=N, esperado=N
{:vector=>[0.00059852403881444408, 0.76646428822591241]}: predicho=S, esperado=N
{:vector=>[0.04752450733408331, 0.423473800874140551]}: predicho=S, esperado=N
{:vector=>[0.12277515842938669, 0.090426390419826471]}: predicho=N, esperado=N
{:vector=>[0.5340275955181082, 0.6610836084556425]}: predicho=S, esperado=S
{:vector=>[0.29979390236963344, 0.6742865461058306]}: predicho=S, esperado=N
{:vector=>[0.09143304434251343, 0.31810846635024226]}: predicho=S, esperado=N
{:vector=>[0.9353816224926145, 0.21867972453231466]}: predicho=N, esperado=N
{:vector=>[0.5728420913802453, 0.8597431704086889]}: predicho=N, esperado=N
{:vector=>[0.8617423354681827, 0.8416350523941717]}: predicho=N, esperado=N
{:vector=>[0.3428911309184238, 0.47248460369197454]}: predicho=N, esperado=N
{:vector=>[0.9344912408283448, 0.6723560017193854]}: predicho=S, esperado=S
{:vector=>[0.353789647179242551, 0.086806765633916651]}: predicho=S, esperado=N
{:vector=>[0.8771039472466586, 0.9189479718790119]}: predicho=N, esperado=N
Resultado: 13/20
```

NSA: Ejemplo

0.846770655	0.96259939	0.603224192	0.643938537	0.045608737	0.740429696	N	N
0.523067415	0.655941239	0.396057941	0.627151752	0.729252323	0.858703218	S	S
0.746587835	0.546581645	0.933282007	0.777823892	0.584872865	0.816942102	S	N
0.709577977	0.938385677	0.079984725	0.042810016	0.022722722	0.591345593	N	N
0.828743734	0.887244372	0.021800041	0.502303058	0.494260586	0.739378834	N	N
0.945526146	0.662767291	0.074926687	0.600001742	0.834949175	0.767479587	S	N
0.563203788	0.551409163	0.77126413	0.22876427	0.378708278	0.34872409	N	N
0.625487291	0.883825725	0.225395625	0.217429731	0.000598524	0.766464288	S	N
0.692292737	0.86051994	0.043638924	0.950822496	0.047524507	0.423473801	S	N
0.545274412	0.63251665	0.094966286	0.057384661	0.122775158	0.09042639	N	N
0.774572944	0.655724611	0.289632148	0.059410148	0.534027596	0.661083608	S	S
0.977472551	0.910199989	0.147895203	0.652857523	0.299793902	0.674286546	S	N
0.674031146	0.737390827	0.679616317	0.630441959	0.091433044	0.318108466	S	N
0.963448226	0.549401619	0.974339871	0.332524744	0.935381622	0.218679725	N	N
0.859596843	0.674585951	0.434786373	0.60920179	0.572842091	0.85974317	N	N
0.545456056	0.521027961	0.520960221	0.278402632	0.861742335	0.841635052	N	N
0.832477459	0.982774912	0.462614995	0.132588927	0.342891131	0.472484604	N	N
0.660112962	0.75374004	0.239449808	0.745212364	0.934491241	0.672356002	S	S
0.751596903	0.526249228	0.096898951	0.190142515	0.353789647	0.086806766	S	N
0.757838267	0.592408396	0.507818544	0.234527706	0.877103947	0.918947972	N	N
		0.125411309	0.942805966				
		0.399665955	0.261137582				
		0.170590829	0.322996334				
		0.213383422	0.206889345				
		0.038146362	0.718098468				
		0.957149508	0.209237074				
		0.277726343	0.13433021				
		0.682522299	0.617819006				



GRACIAS

Dr. Edward Hinojosa Cárdenas
ehinojosa@unsa.edu.pe