Capítulo 4 Algoritmos de aproximación

4.1 Fundamentos de los Algoritmos de Aproximación

Muchos problemas NP-Completos son formulados en forma natural como problemas de optimización. Se sabe que la programación dinámica y los algoritmos de Ramificación y Acotamiento (Branch and Bound) que son algoritmos exactos para encontrar soluciones óptimas requieren tiempo superpolinomial a pesar de que estos algoritmos son más eficientes que la búsqueda exhaustiva pura, su tiempo de corrida es aún exponencial. Por esa razón es entonces natural buscar otra alternativa de solución como son los algoritmos de aproximación que trabajan en tiempo polinomial [YAN94], [MEH93].

Definición 4.1 : *La clase de problemas de optimización que se derivan de problemas de decisión en NP es llamada NPO, como un acrónimo para designar que son problemas de optimización derivados de problemas NP* [DJO74].

Así, los algoritmos de aproximación surgen de la inquietud de la comunidad científica en los años 70's, donde se pensó que las dos clases P y NP eran distintas y que por lo tanto no había algoritmos de tiempo polinomial que fuesen capaces de resolver estos problemas en NPO. Sin embargo dado que existían muchos problemas que tenían fuertes aplicaciones en la práctica era importante determinar la tratabilidad computacional de ellos, por lo que muchos investigadores comenzaron a considerar relajaciones de estos problemas de manera que se obtuvieran versiones que fueran tratables. Así nace la primera relajación considerada que fue la condición de "optimabilidad", por lo que se enfocaron a la construcción de algoritmos de aproximación que trabajan en tiempo polinomial y que calculan soluciones cercanas al óptimo global en un tiempo pequeño; es decir, soluciones que garantizaran estar dentro de un factor multiplicativo de la solución óptima [DJO74], [MEH93]. De lo que se deriva la definición de un algoritmo de aproximación:

Definición 4.2 : *Un algoritmo A es un algoritmo de aproximación para un problema de optimización $\tilde{O}$ en NPO si dada una instancia de entrada I, éste calcula una solución factible S para cada entrada I* [JOD74].

Lo interesante de la solución de estos problemas es precisamente que si son algoritmos de aproximación, ¿qué tanto se aproximan a su solución real y cuál es un valor garantizado de exactitud?, para ello existe una métrica teórica estándar para medir la calidad de un algoritmo de aproximación, que es el cociente para el peor de los casos que indica que tan cerca del valor óptimo están las soluciones encontradas por un algoritmo de aproximación. Dicha cercanía puede ser expresada como el cociente del valor obtenido por el algoritmo de aproximación sobre el valor de la solución óptima. Se usará la notación OPT(I) para denotar el valor óptimo de la función objetivo en la instancia I y V(I,A(I)) para denotar el valor obtenido por el algoritmo A al evaluar la instancia I en la función objetivo [JOD74].

Definición 4.3 : *Se define el cociente de aproximación de un algoritmo A, denotado por C_A, para una instancia I de un problema de optimización $\tilde{O}$(con $|I| = n$) como* [JOD74] :

$$C_A = \min\left\{\frac{V(I,A(I))}{OPT(I)}, \frac{OPT(I)}{V(I,A(I)}\right\}$$

Nótese que C_A es siempre menor o igual a 1 y el factor de garantía r_A del algoritmo A para Π será :

$$r_A = \min \{C_A \mid \text{para toda I de } \Pi \}$$

A una solución que está dentro de un factor multiplicatico r_A del valor óptimo se le conoce como una r_A-aproximación, y decimos que :

Definición 4.4 : *Un problema NPO es aproximable dentro de un factor r_A si éste tiene un algoritmo de aproximación de tiempo polinomial con factor de garantía r_A* [JOD74].

Se dice que un algoritmo A es un algoritmo de aproximación-α para un problema de optimización Π, con α una constante, Si A es un algoritmo de aproximación tal que para toda instancia I de Π, produce una solución que está dentro de α-veces el OPT(I). También es usual considerar el término algoritmo de aproximación-α, para algoritmos aleatorios de tiempo polinomial que proporcionan soluciones cuyo valor esperado es al menos α-veces el óptimo. A α se le llama la constante de aproximación o bien la eficiencia garantizada que proporciona el algoritmo A. Es por eso que los problemas se clasifican según su factor de aproximación en [YAN94]:

a) Problemas que no pueden aproximarse dentro de ningún factor constante α.

b) Problemas de optimización que se ha demostrado poseen un factor constante de aproximación; y que a pesar del trabajo realizado, no se han podido mejorar tales factores [JOD74], [YAN94].

Definición 4.5 *Un problema $\tilde{O}$ esta en la clase APX (Problema de Aproximación X) si existe un algoritmo de tiempo polinomial para $\tilde{O}$ cuyo factor de aproximación o garantía está acotado por una constante* [DJO74].

Por ejemplo, dentro de los problemas NP-Completos se encuentra el PAV, que cuando el costo de las aristas satisface la desigualdad del triángulo (Δ PAV) existe un algoritmo simple que, siempre produce una ruta de longitud a lo mas dos veces la longitud de la ruta óptima. Esto tiene un tiempo de ejecución de $O(n^2)$. Un algoritmo mejorado con tiempo de corrida de $O(n^4)$ siempre encuentra una ruta con constante de aproximación de a lo más 3/2 de longitud de la ruta óptima. Y hasta la fecha no se conoce una mejor aproximación del algoritmo [MEH93].

Sea f una función, f-APX denota la clase de problemas en NPO que son aproximables dentro de un factor f, así, se obtiene una jerarquía de clases de complejidad.

Por ejemplo, poly-APX y log-APX son las clases de problemas en NPO los cuales tienen, respectivamente, algoritmos de aproximación con un factor de aproximación o garantía acotado polinomialmente y logarítmicamente, con respecto a la longitud de la entrada [DJO74].

c) A la clase de los problemas que pueden aproximarse tanto como se quiera a cualquier factor constante ε arbitrariamente cercano a uno usando algoritmos con complejidad polinomial en tiempo se les dice que tienen un esquema de aproximación de tiempo polinomial PTAS (por sus siglas en inglés, Polynomial time approximation scheme) [YAN94].

Definición 4.6 : *Un problema Õ esta en la clase PTAS (Problema de Aproximación de tiempo polinomial) si para cualquier racional $e > 0$, existe un algoritmo de aproximación de tiempo polinomial para Õ cuyo factor de garantía está acotado por $(1+ e)$* [DJO74].

Con esto como antecedente, se puede ver que en la Naturaleza existen muchos procesos que buscan un estado estable. Dichos procesos pueden contemplarse como procesos naturales de optimización. En los últimos años se han propuesto varios algoritmos de optimización global que simulan estos procesos naturales de optimización. Tres ejemplos clásicos, son [WWW9] :

a) Recocido Simulado (Simulated Annealing), basado en el proceso natural de enfriamiento de un sólido hasta que alcanza su punto de equilibrio.

b) Algoritmos Evolutivos, basados en el concepto de evolución biológica. Entre los mismos se pueden considerar las siguientes ramas : Algoritmos Genéticos, Estrategias Evolutivas, Programación Evolutiva y Programación Genética.

c) Redes Neuronales Artificiales, basadas en procesos relacionados con el sistema nervioso central.

En este documento se hará una breve descripción solo de los algoritmos Genéticos, Dos Optimal (2-Opt), Adaptación Prim, Híbrido Dos Optimal-Prim y el de Redes Neuronales (Red de Hopfield).

4.2 Fundamentos de los Algoritmos Genéticos

La naturaleza nos enseña grandes maravillas, así los Algoritmos genéticos AGs (Genetic Algorithms) son una analogía de la estructura genética, es decir, basándose en los principios de selección natural se realizan procedimientos que simulan el comportamiento de los cromosomas dentro de una población, éste método ha resultado ser muy eficiente para buscar aproximaciones a mínimos globales en espacios grandes y complejos en relativamente poco tiempo [MOR91].

Los componentes básicos de un algoritmo genético son [WWW2]:

1. Operadores genéticos (mutación y cruzamiento)
2. Una representación apropiada del problema a resolver
3. Una función de Oportunidad o bien llamada de Aptitud (fitness)
4. Un procedimiento de inicialización

El método AGs como preámbulo utiliza el procedimiento de inicialización para generar la primera población. Los miembros de la población a semejanza con los cromosomas son usualmente cuerdas de símbolos que permiten representar las posibles soluciones al problema que está tratando de ser resuelto. Simulando a los individuos de la población, estos compiten unos contra otros para ser usados en el proceso de reproducción. Es decir, cada uno de los miembros de la población es evaluado, y según su "Aptitud (fitness)" se le asigna una probabilidad de ser seleccionado para la reproducción. Matemáticamente hablando se usa esta probabilidad para que los operadores genéticos seleccionen algunos individuos, obteniendo de esta forma a nuevos individuos, donde generalmente son aquellos individuos con más éxito en cada competencia y que serán capaces de producir mayor descendencia que los que no tienen éxito [WWW2].

El operador de cruzamiento de entre todos los individuos selecciona dos miembros de la población y combina sus cromosomas para generar a los mejores hijos. Se puede decir apoyando lo anterior que los genes de individuos buenos se propagan con más probabilidad a lo largo de la población, por esa razón dos padres buenos producirán alguna descendencia que se espera será mejor que la de sus antecesores. El operador de mutación selecciona un miembro de la población y modifica alguna parte de su cromosoma. Los elementos con peor "Aptitud (fitness)" son reemplazados por los nuevos individuos. Se espera que cada generación sucesiva se adapte mejor a su medio ambiente [WWW3].

4.2.1 Funcionamiento de un Algoritmo Genético

Los algoritmos Genéticos son diferentes a los métodos tradicionales como se describe en los siguientes cuatro puntos [WWW3] :

a) Trabajan codificando un conjunto de parámetros.
b) Buscan dentro de una población de puntos, no en un simple punto.
c) Usan una función objetivo que da información de rentabilidad y no se deriva de otros conocimientos.
d) Usan reglas de transición probabilisticas, no reglas deterministicas.

La función de aptitud no es más que la función objetivo de nuestro problema de optimización. El algoritmo genético únicamente maximiza, pero la minimización puede realizarse fácilmente utilizando el recíproco de la función maximizante (debe cuidarse, por supuesto, que el recíproco de la función no genere una división entre cero). Una característica que debe tener esta función es que tiene que ser capaz de "castigar" a las malas soluciones, y de "premiar" a las buenas, de forma que sean estas últimas las que se propaguen con mayor rapidez [WWW3].

Los algoritmos Genéticos requieren de un conjunto de parámetros naturales de optimización que serán codificados como una cadena de longitud finita sobre algún alfabeto finito. La codificación más común de las soluciones es a través de cadenas binarias, aunque se han utilizado también números reales y letras. El primero de estos esquemas ha gozado de mucha popularidad debido a que es el que propuso originalmente Holland, y además porque resulta muy sencillo de implementar [WWW3].

Este método tiene ventajas sobre otras técnicas de búsqueda.

a) No necesitan conocimientos específicos sobre el problema que intentan resolver.

b) Operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales.

c) Cuando se usan para problemas de optimización (maximizar una función objetivo), resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales.

Pero como desventaja pueden converger prematuramente debido a una serie de problemas de diversa índole.

4.2.2 Alcances y Limitaciones de los Algoritmos Genéticos

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia. El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los Algoritmos Genéticos [WWW9].

4.2.3 Algoritmo Genético Simple

El Algoritmo Genético Simple, también denominado Canónico, se representa en la Figura 4.1. Como se verá a continuación, se necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de ajuste ó adaptación al problema, la cual asigna un número real a cada posible solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las funciones anteriores será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población [WWW9].

```
{01} Algoritmo_Genético_Simple;
{02}  Begin
{03}     Generar una población Inicial;
{04}     Calcular la función de evaluación de cada individuo;
{05}     While NOT Terminado do
{06}        Begin  // Producir una nueva generación
{07}           For Tamaño_población /2 do
```

```
{08}          Begin // Ciclo Reproductivo
{09}               Seleccionar dos individuos de la generación anterior, para el cruce
                   (probabilidad de selección proporcional a la función de evaluación
                   del individuo);
{10}               Cruzar con cierta probabilidad los dos individuos obteniendo dos
                   descendientes;
{11}               Mutar los dos descendientes con cierta probabilidad;
{12}               Calcular la función de evaluación de los dos descendientes
                   mutados;
{13}               Insertar los dos descendientes mutados en la nueva generación;
{14}          End del for;
{15}        If la población ha convergido
{16}          then
{17}            Terminado = True;
{18}        end del if
{19}      end del while
{20}  end del Algoritmo_Genético_Simple;
```

Figura 4.1 Pseudocódigo del Algoritmo Genético Simple

4.2.4 Codificación

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que se denominan penes), los cuales agrupados forman una ristra[1] de valores (a menudo referida como cromosoma). Si bien el alfabeto utilizado para representar a los individuos no debe necesariamente estar constituido por el {0, l), buena parte de la teoría en la que se fundamentan los Algoritmos Genéticos utiliza dicho alfabeto. En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina *fenotipo*. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como genotipo [WWW9]. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta última puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación. La función de adaptación debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de adaptación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma. Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La selección de padres se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación [WWW9].

Este procedimiento se dice que está basado en la ruleta sesgada. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que, los pobremente adaptados al problema, no se escogerán más

[1] Una ristra es una cadena de caracteres formada a partir de elementos del alfabeto ampliado S', siendo S' = (0,1})

que de vez en cuando. Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los operadores de cruce y mutación. Las formas básicas de dichos operadores se describen a continuación. El operador de cruce, toma dos padres seleccionados y corta sus *ristras* de cromosomas en una posición escogida al azar, para producir dos subristras iniciales y dos subristras finales. Después se intercambian las subristras finales, produciéndose dos nuevos cromosomas completos, véase la Figura 4.2 [WWW9].

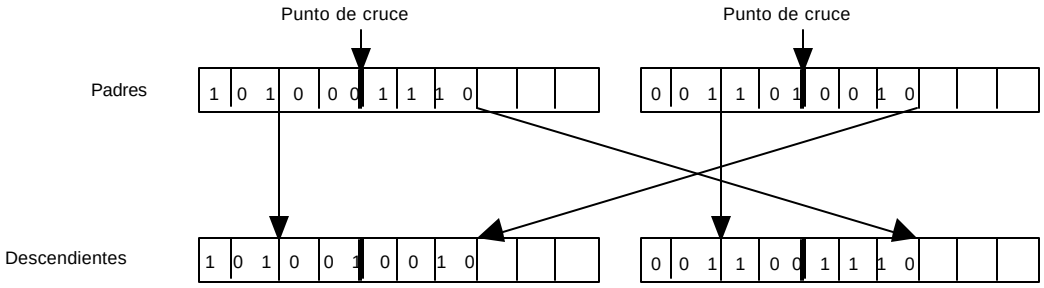


Figura 4.2 Operador de cruce basado en un punto

Ambos descendientes heredan genes de cada uno de los padres. Este operador se conoce como *operador de cruce* basado en un punto. Habitualmente el operador de cruce no se aplica a todos los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1.0. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando a los padres [WWW9]. El operador de mutación se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. La Figura 4.3 muestra la mutación del quinto gen del cromosoma.

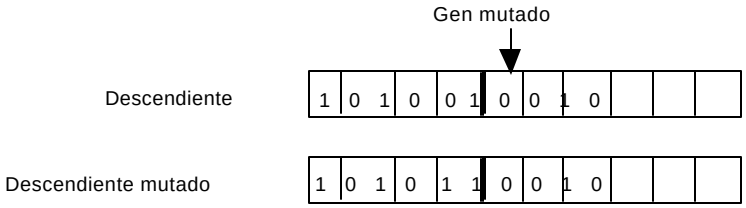


Figura 4.3 Operador de mutación

Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, éste último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los Algoritmos Genéticos. Para criterios prácticos, es muy útil la definición de convergencia introducida en este campo por De Jong en su tesis doctoral (K.A. 1975, An analysis of the behaviour of a class of genetic adaptive systems, Tesis doctoral, University of Michigan). Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global [WWW9].

El concepto de convergencia está relacionado con la progresión hacia la uniformidad: se asevera que un gen ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en que al menos algunos individuos de la población hayan convergido. A medida que el número de generaciones aumenta, es más probable que la adaptación media se aproxime a la del mejor individuo [WWW9].

4.3 Método Dos Optimal

Esta técnica de solución también recibe el nombre de "2-Opt" que es la forma corta de "Dos Optimal", o bien "2-Operadores" u " Opción Doble". Esta es una de las heurísticas mas exitosas para obtener una solución cercana al problema PAV [AHO74]. Este inicia con un ciclo Hamiltoniano llamado H, esta ruta inicial para el PAV puede ser arbitraria o bien (1, 2, …, n). Se borran r aristas de H, produciendo así r caminos desconectados (algunos de los cuales pueden ser nodos aislados). Se reconectan estos r caminos de tal forma que producen otra ruta para el PAV llamada H', en la que se usan aristas diferentes de aquellas que fueron removidas de H. Así H y H' difieren una de la otra en exactamente r aristas; las (n-r) aristas remanentes son las aristas en común [ARR96]. La siguiente figura ilustra el método 2-Opt.

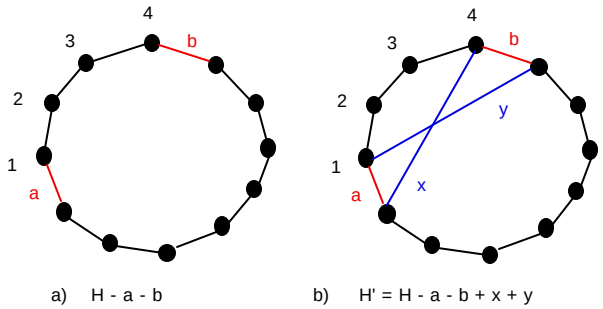


Figura 4.4 Unión de vértices para formar un camino 2-Opt

Se calcula el peso total w(H') de la ruta H', y si w(H') < w(H), reemplazar H con H' y repetir el proceso; de otra manera, tomar otro conjunto de r aristas de H para intercambiarlas. Tales intercambios (de conjuntos de r aristas) se continuan hasta que ninguna otra mejora se pueda realizar por intercambio de r aristas. La solución final es llamada r-Optimal o r-Opt, en este caso se estudiará con r = 2 siendo así el Dos-Optimal o 2-Opt [ARR96]. El procedimiento de intercambio de aristas terminará en un óptimo local (no necesariamente en un óptimo global), produciendo de esta manera una solución aproximada. Este algoritmo de aproximación para el PAV ilustra una aproximación para resolver muchos problemas de optimización, que se conocen como búsqueda local o búsqueda por vecindades. Este método de mejoras sucesivas en la ruta del PAV puede ser usada para PAV Simétricos tanto como para Asimétricos. Para los PAV Simétricos con número de nodos n • 5, el valor de r puede variar desde 2 a n, sin embargo para los PAV Asimétricos r no puede ser menor que 3 [ARR96].

En general, entre mas alto sea el valor de r en el procedimiento de intercambio, mejor será la solución. Pero el gasto computacional también se incrementa rápidamente con el valor de r. Hay $\binom{n}{r}$ subconjuntos de r aristas en un ciclo con n aristas. Intercambiar cada uno de estos pares de aristas una vez requiere tiempo $O(n^r)$. Por lo que se debe hacer conciencia de lo que se desea, es decir entre la exactitud de la solución y el gasto computacional. Si r es dos se trata del método 2-Opt y tendrá por lo tanto un $O(n^2)$ [ARR96]. En este trabajo se discutirá el 2-Opt porque da muy buenas aproximaciones, particularmente si la ruta inicial es buena [AHO74].

El procedimiento 2-Opt es una solución aproximada al problema Simétrico del PAV con n nodos y una matriz de adyacencia o de pesos W. El ciclo inicial consiste del siguiente conjunto de aristas (de un PAV Simétrico) del $H = (x_1, x_2, ..., x_n)$ en el orden $x_1, x_2, ..., x_n$. Sea $X = \{x_i, x_j\}$ el conjunto de dos aristas en H las cuales serán borradas y reemplazadas por las aristas $Y = \{y_p, y_q\}$, si hay una mejora. Esto es $H' = (H - X) \cup Y$ es una ruta nueva y mejorada. Obsérvese que [ARR96] :

a) las dos aristas x_i, x_j en X no pueden ser adyacentes y

b) Una vez que el conjunto X ha sido seleccionado, el conjunto Y es determinado.

Así hay :

$$\frac{n(n-1)}{2} - n = \frac{n(n-3)}{2}$$

rutas posibles H' para una H dada. Para cualquiera de estas rutas se denotará la mejora con δ, así :

$$\delta = w(H) - w(H') = w(x_i) + w(x_j) - w(y_p) - w(y_q)$$

Se examinarán todas las n(n-3)/2 rutas H' y se retendrá una de ellas, aquella cuya δ sea la máxima. Si esta δ_{max} es negativa o cero, se ha encontrado la solución 2-Opt. Si $\delta_{max} > 0$, se usa la correspondiente solución como la ruta inicial y entonces se repite todo el proceso. Se continua sucesivamente la mejora de la ruta hasta que δ_{max} es un número negativo [ARR96].

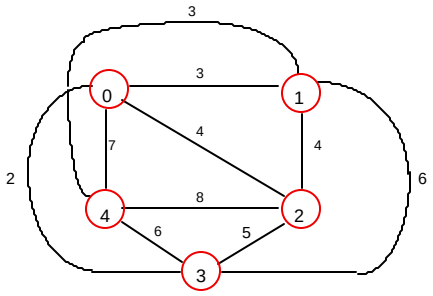
El siguiente algoritmo describe a 2-Opt como técnica de solución para el PAV, usando reemplazamientos de dos aristas en forma sucesiva [ARR96].

```
{01} Procedure 2-Opt
{02}   begin
{03}     Sea H = (x1, x2, ....., xn) la ruta inicial
{04}     repeat
{05}       δmax ← 0;
{06}       for i ← 1 to (n-2) do
{07}         for j ← (i+2) to n or (n-1) do
{08}           begin
                 // el último caso es solamente cuando i = 1
{09}             if (w(xi) + w(xj)) - (w(yp) + w(yq)) > δmax
{10}               then
                     // se ha encontrado un mejor par de aristas
```

```
{11}                    begin
{12}                      δmax ← (w(xi) + w(xj) ) - (w(yp) + w(yq));
{13}                      guardar tanto a i como a j
{14}                    end del if;
{15}              end del for j;
{16}           if δmax > 0
{17}             then
{18}               begin
                     // intercambia un par de aristas, porque la ruta es ahora mas corta
{19}                   H = H - {xi, xj} ∪ {yp, yq}
{20}               end del if;
{21}       until δmax = 0;
{22} end del Procedure 2-Opt
```

Figura 4.5 Algoritmo 2-Opt

Usando el algoritmo anterior, se realiza una descripción más detallada del 2-Opt para un PAV Simétrico de 5 ciudades. Se muestra el PAV en la figura 4.6a con su respectiva matriz de adyacencia [AHO74].



a) Grafo

i/j	0	1	2	3	4
0	0	3	4	2	7
1	3	0	4	6	3
2	4	4	0	5	8
3	2	6	5	0	6
4	7	3	8	6	0

b) Matriz de Adyacencia

Figura 4.6 Un problema PAV Simétrico de 5 ciudades

Con los datos de la matriz de adyacencia se calcula el circuito Hamiltoniano inicial (ruta inicial) a mejorar que es 0-1-2-3-4-0 con un peso de 3+4+5+6+7 = 25. Ahora, reconsidérese la figura 4.6, para comenzar con el recorrido de la figura 4.7.a. El proceso se inicia con $\delta_{max} = 0$ y con la selección de 2 aristas; la formada por los vértices (0, 1) y una arista vecina (2, 3) (figura 4.7.a). Se proponen dos aristas nuevas, las formadas por el cruce de (i_1, j_1), y la otra por el cruce de (i_2, j_2), cuyas distancias son :

Distancia (i_1, i_2) = Distancia (0, 1) = 3
Distancia (j_1, j_2) = Distancia (2, 3) = 5
Distancia (i_1, j_1) = Distancia (0, 2) = 4
Distancia (i_2, j_2) = Distancia (1, 3) = 6

Se calcula su peso :
max1 = (a[i1][i2] + a[j1][j2]) - (a[i1][j1] + a[i2][j2]) = -2
max1 no es mayor que δ_{max} así que se buscan otras 2 parejas.

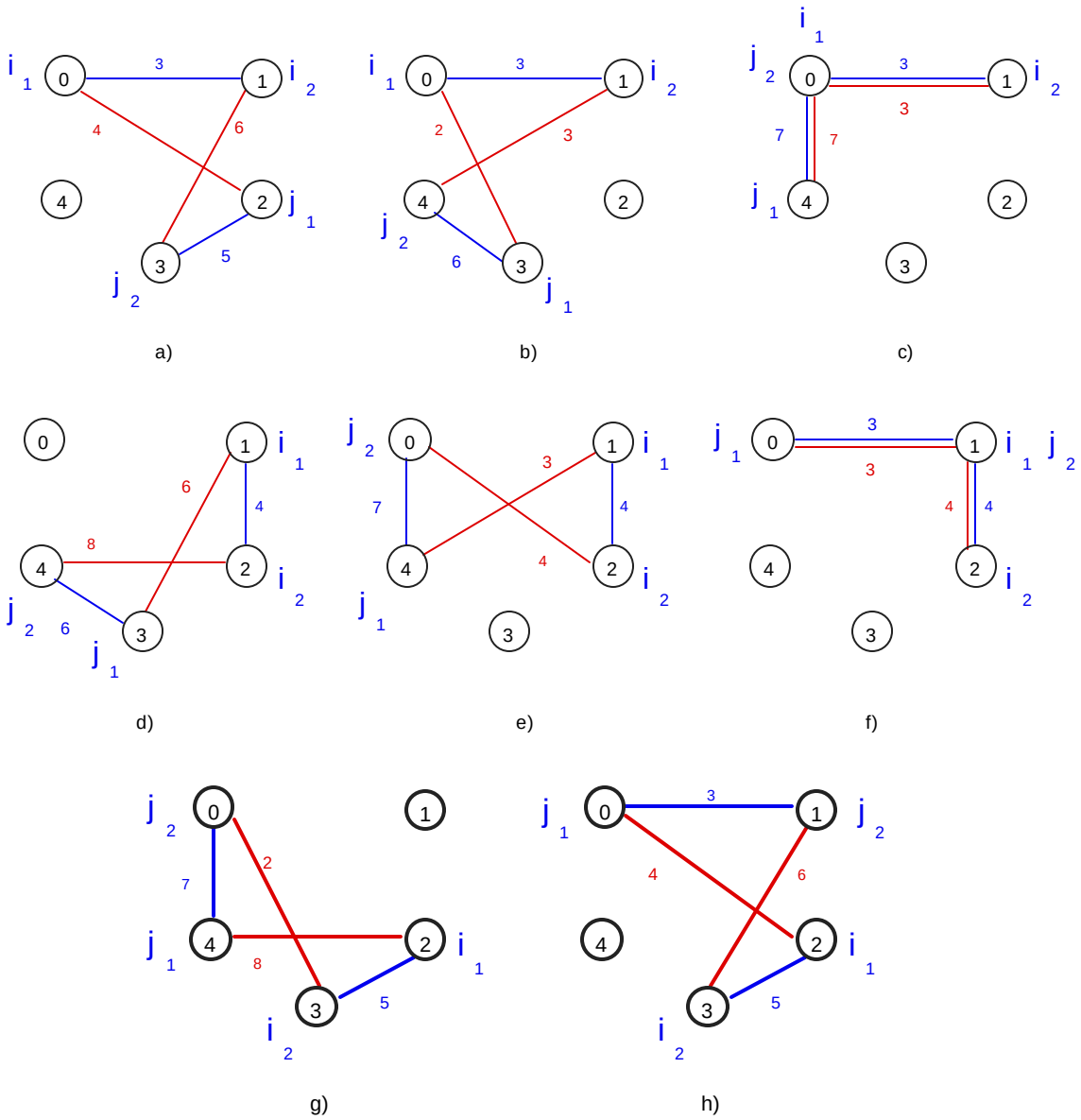


Figura 4.7 Aristas mejoradas de un problema PAV Simétrico de 5 ciudades

Se reemplazan (i_1, i_2) y (j_1, j_2), cuyo costo total es 9, por (i_1, j_1) y (i_2, j_2), con un costo total de 5; y una diferencia (3+6) - (2+3) = 4, como se muestra en la figura 4.7.b. Dando un peso actual de 25 (peso anterior) - 4(mejora) = 21. A pesar de que el autor dice que las aristas vecinas no deben ser adyacentes, hay al final de cada iteración una adyacente como se muestra en la figura 4.7.c y 4.7.f. Se continua el proceso de búsqueda de mejores aristas, encontrando otra mejora como se muestra en la siguiente figura [AHO74].

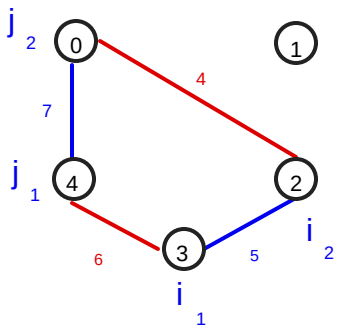


Figura 4.8 Otro par de aristas mejoradas

Esta vez se reemplazan (i_1, i_2) y (j_1, j_2), cuyo costo total es 12, por (i_1, j_1) y (i_2, j_2), con un costo total de 10; y una diferencia 2, como se muestra en la figura 4.8. Dando un peso actual de 21 (peso anterior) - 2(mejora) = 19. Se puede comprobar que no hay ningún par de aristas que se pueda eliminar de esta figura y ser reemplazado con ventaja por aristas cruzadas con los mismos extremos. Por lo que el peso es el peso mínimo, dando el recorrido óptimo mostrado en la figura 4.9.a, esta ruta se redibuja en la figura 4.9.b siguiendo los nodos para que sea mas entendible. La ruta tomando a la ciudad 0 como la inicial es 0-3-4-1-2-0 con un peso óptimo de 19 [AHO74].

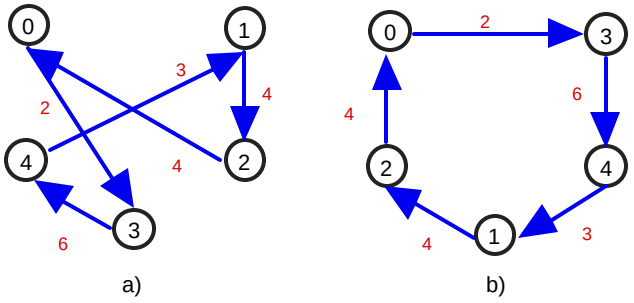


Figura 4.9 Ruta óptima después de 2-Opt

Es fácil verificar que, para una k fija, el número de diferentes transformaciones con opción k que se necesita considerar, si existen n vértices, es $O(n^k)$. Por ejemplo, el número exacto es n(n-3)/2 para k =2. Sin embargo, el tiempo requerido para obtener un recorrido localmente optimal puede ser mucho mayor que esto, ya que se pudieron haber realizado muchas transformaciones locales antes de alcanzar un recorrido localmente óptimo, y cada transformación de mejora introduce aristas nuevas que pueden participar en transformaciones posteriores que mejoran aún más el recorrido fijado. Lin y Kernighan en 1973 encontraron que la opción de profundidad variable es un método muy poderoso en la práctica, y tiene una buena oportunidad de obtener el recorrido óptimo en problemas de ciudades entre 40 y 100 [AHO74].

4.4 Método Adaptación Prim

El método Adaptación Prim se adecuó al PAV, porque el Algoritmo de Prim, produce un árbol generador mínimo o árbol de expansión mínimo o árbol abarcador

mínimo T (Minimum spanning tree) que es un grafo no dirigido conectado con aristas ponderadas (ver Capítulo 2). Inicialmente se escoge un vértice arbitrario y se permite que T sea un árbol que consiste de ese vértice seleccionado. Se repite la etapa de selección del siguiente vértice n-1 veces, tomando en cuenta cual arista tiene el costo mínimo con exactamente un punto final en T, incluyendo ese vértice en T [KOZ91]. Se dice que es un algoritmo voraz porque siempre añade la arista de costo mínimo y suprime la mayor. Además hace elecciones mínimas manteniendo simultáneamente el subgrafo conexo y acíclico. Más aún, no necesita que las aristas del grafo *G* estén ordenadas de antemano [KEN90]. La adaptación consiste en tomar los vértices del árbol abarcador mínimo y producir un ciclo hamiltoniano.

Sea *Best_S_so_Far* el conjunto de vértices en el momento en el que el algoritmo se detiene. Sean V(T) los vértices del subgrafo *T* (árbol de expansión mínimo). Sean *w* el peso de la suma de las aristas ponderadas del subgrafo T y *a* la matriz de adyacencia del grafo G. El procedimiento forma un árbol abarcador mínimo. En cada etapa el algoritmo busca una arista de menor peso que una un vértice en T con un nuevo vértice fuera de T. Entonces añade esa arista y ese vértice a T y repite el proceso [KEN90]. El algoritmo se describe a continuación :

```
{01} Function Adaptación_Prim
{02}   begin
{03}     Se inicializa el vector col a ceros;
{04}     menor_en_fil ← •;
{05}     dir ← 0;
{06}     i ← 0;
{07}     Best_S_so_Far[0] ← 0;
{08}     col[0] ← 1;
{09}     for j ← 1 to n_vertices - 1 do
{10}       begin
{11}          menor_en_fil ← •;
{12}          for k ← 0 to n_vertices -1 do
{13}               begin
{14}                 if (col[k] = 0) and (menor_en_fil > a[Best_S_so_Far[i]][k])
{15}                    begin
{16}                       menor_en_fil ← a[Best_S_so_Far[i]][k];
{17}                       dir ← k;
{18}                     end del if;
{19}             end del for k;
{20}          Best_S_so_Far[j] ←dir;
{21}          col[Best_S_so_Far[j]] ← 1;
{22}          i ← j;
{23}         w ← w + menor_en_fil;
{24}       end del for j
{25}    w ← w + a[Best_S_so_Far[n_vertices-1]][0];
{26}   return w;
{27} end de la function Adaptación_Prim
```

Figura 4.10 Algoritmo Adaptación_Prim

Se explicarán las líneas más importantes :

{03} Se lleva un registro para cada vértice *x* en *V(G)* del vértice *u* en *a* con menor valor *a(u,x)* en el vector *col* que es inicializado en ceros. Cuando se localice un vértices adecuado la posición correspondiente se marcará con 1.

{04} y {10} Para localizar el menor de cada fila de la matriz de adyacencia se inicializa el menor_en_fil en •.

{07} Para ir guardando la ruta se usa el vector Best_S_so_Far que es inicializado con el nodo 0 en la dirección 0;

{08} El vector col es marcado con 1 porque ya se tomo el nodo 0 como punto de partida.

{09} El ciclo for corre desde 1 hasta n_vertices - 1 porque ya se tomo el nodo 0 anteriormente.

{12} a {19} Se busca la arista con menor valor en peso.

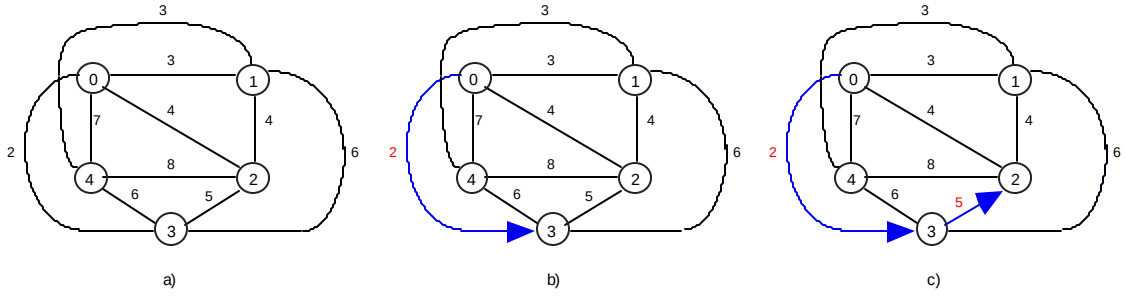
{20} a {21} se localiza la dirección de esa arista y se guarda en el vector Best_S_so_Far marcando al vector col con 1 en la dirección de Best_S_so_Far[j].

{23} Se actualiza el peso sumándole a este el menor_en_fil localizado.

{25} Aquí se adapta para que se cierre el camino Hamiltoniano, sumando al peso el peso de la matriz en la dirección de Best_S_so_Far[n_vertices-1]][0].

El tiempo para encontrar la x más cercana en *a* y después actualizar registros es O(n). Pero finalmente en el peor de los casos corre en un tiempo $O(n^2)$ por los dos ciclos anidados [KEN90].

Usando el algoritmo anterior, se realiza una descripción más detallada del método Adaptación Prim para el PAV Simétrico de 5 ciudades que se muestra en la Figura 4.6 [AHO74]. Se parte de la Figura 4.11.a, puede iniciarse en cualquier nodo, pero en este caso será el nodo 0. Se busca la arista con el menor peso que salga del nodo 0, siendo el nodo 3 con un peso de 2. Y se repite la operación, recordando que un nodo que ya fue seleccionado no puede participar nuevamente. Como se muestra en la Figura 4.11.a a Figura 4.11.g.



peso = 0 **peso = 2** **peso = 2 + 5**

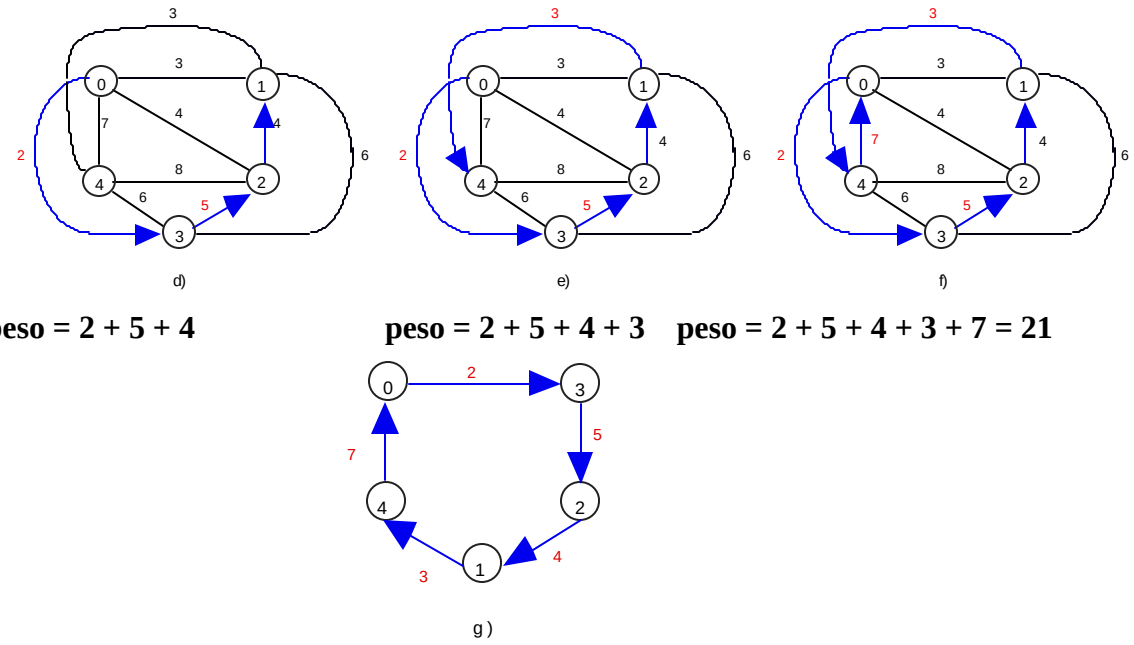


Figura 4.11 Ruta óptima después de Adaptación_Prim

En la Figura 4.11.d hay dos números de peso mínimo con valor 4, pero el nodo 0 no puede seleccionarse porque primero formaría un ciclo lo cual no es permitido en el árbol abarcador de Prim, y segundo se repetiría el nodo 0 cosa no permitida en el PAV. Por lo que solo queda el mínimo 4 que va al vértice 1.

En la Figura 4.11.f se encuentra el peso mínimo y la ruta óptima, esta se reacomoda para que se distinga mejor en la figura 4.11.g. Además puede notarse que el resultado de la ruta óptima es 21 que es una aproximación al valor real que es de 19.

4.5 Método Híbrido Dos Optimal-Prim

La idea de la hibridación surgió porque la teoría de Dos-Optimal dice que este método es mejor cuando se le da una ruta inicial buena [AHO74]. Así que se uso la Adaptación de Prim para encontrar la ruta inicial y después esta ruta es usada por el método Dos-Optimal, dando así el método Híbrido Dos optimal-Prim. No se describe ningún algoritmo porque los descritos con anterioridad fueron empleados como un llamado a la función Adaptación Prim seguida del llamado a Dos-Optimal.

4.6 Redes Neuronales Artificiales

4.6.1 Conceptos generales

Las redes neuronales artificiales (RNA) o modelos conexionistas constituyen un modelo de computación y por lo tanto suministran un método para la solución de problemas por computadora. Característico de este modelo es la distributividad del conocimiento, su procesamiento y el paralelismo de los procesos en la solución de problemas. Ha habido inspiración biológica en el diseño de los modelos conexionistas partiendo de la evidencia anatómica y fisiológica que existe para considerar a nuestro

cerebro como un modelo de computación paralelo-distribuido [GAG99]. Hay alrededor de 10 billones (10^{11}) de neuronas en el cerebro humano. Una neurona recibe entradas de otras células, es decir se comunica con otras neuronas y genera a su vez salidas que recepcionan otras neuronas u órganos efectores como los músculos y las glándulas. Las neuronas reciben entradas de otras neuronas a través de conexiones llamadas *dendritas* y envían sus salidas a otras neuronas mediante una conexión especial denominada *axón*. Las dendritas reciben señales de otras células en puntos de conexión llamadas *sinapsis* [GOG99], [GAG99]. De acá, las señales se pasan al cuerpo de la célula, donde son esencialmente "promediadas" con otras señales. Si este promedio en un determinado tiempo es suficientemente grande, la célula es excitada, mandando un pulso a través del axón a otras células [GOG99]. Una neurona puede recepcionar una enorme cantidad de entradas y puede enviar su salida también a un número considerable de neuronas [GAG99].

Desde un punto de vista fisiológico, una neurona es un sistema electroquímico complejo provisto de un potencial continuo interno denominado *potencial de membrana*. Cuando el potencial de membrana excede un *umbral*, la neurona puede producir y enviar como salida a través de su axón lo que se denomina un *potencial de acción* todo-o-nada. A expensas de los potenciales de acción que concurren en la sinapsis tienen lugar dos formas fundamentales de sinapsis : la *sinapsis excitatoria* , que puede dar lugar en una neurona a un potencial de acción, y la *sinapsis inhibitoria* , que puede disminuir la posibilidad de que una neurona produzca un potencial de acción. En el desarrollo de los modelos conexionistas no hay siempre empeño en lograr "plausibilidad biológica" (modelar el cerebro), sino en desarrollar modelos bajo la concepción general paralelo-distribuída : la neurona típica o unidad de una red neuronal artificial (RNA) es una abstracción e idealización de rasgos estructurales y funcionales de la neurona biológica, ver figura 4.12 [GAG99].

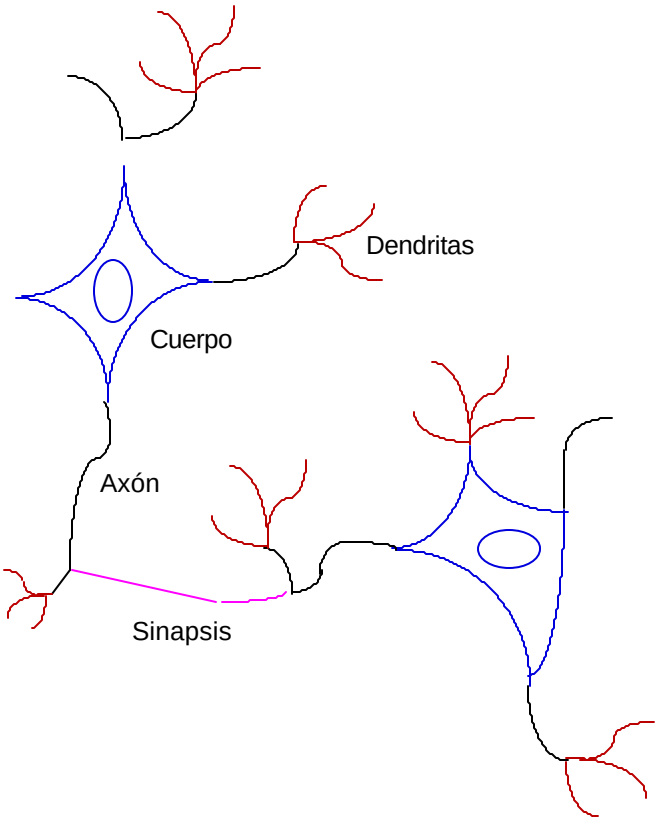


Figura 4.12 Componentes de una neurona biológica

Nacen así las redes neuronales artificiales (RNA) que son modelos matemáticos inspirados en sistemas biológicos. Las RNA se conocen con diferentes nombres, como por ejemplo modelos conexionistas (connectionist models), procesamiento distribuido en paralelo (Parallel Distributed Processing ó PDP), Sistemas Neuronales Artificiales (Artificial Neural Systems ó ANS), sistemas neuromórficos, etc. [GOG99], [ZUR92].

4.6.2 Características de las RNA

Las RNA semejan en su comportamiento a las funciones más elementales de una neurona biológica. Estos elementos se organizan de una manera que puede o no estar relacionada a la manera en que está organizado el cerebro [GOG99].

No obstante su burdo parecido a los sistemas biológicos, las RNA presentan algunas características propias del cerebro como son:
a) Pueden aprender de la experiencia.
b) Pueden generalizar a partir de ejemplos previos.
c) Pueden abstraer características esenciales de entradas que contienen datos irrelevantes.

Cabe recalcar que las RNA matemáticamente hablando no modelan fielmente las funciones cerebrales, dado que el cerebro humano es muy difícil de modelar y posiblemente nunca se logre. Las características más importantes de las RNA son [GOG99] , [ZUR92] :

a) Habilidad de Aprendizaje

Las RNA pueden aprender modificando su comportamiento en respuesta al medio ambiente, dado que se auto-ajustan produciendo respuestas consistentes con el medio ambiente. Existen muchos algoritmos de aprendizaje que pueden aplicarse a las RNA.

b) Generalización

Una vez que ha aprendido se dice que la RNA ha sido entrenada, y entonces ésta es, hasta cierto grado, insensible a variaciones pequeñas en sus entradas. Por lo que puede decirse que las RNA producen sistemas capaces de manejar el mundo "imperfecto" en que vivimos. Nótese que las RNA generalizan automáticamente como resultado de su estructura y no debido a inteligencia humana "introducida" en forma de programas "ad-hoc", como sucede en el caso de los Sistemas Expertos.

c) Abstracción

Algunas RNA al entrenarse son capaces de abstraer la esencia de una serie de entradas, dando como resultado que pueden abstraer patrones perfectos de modelos distorsionados.

4.6.3 Aplicaciones de las RNA

Las RNA no son capaces de resolver todo tipo de problema y por tanto no son el sustituto de las computadoras convencionales; por ejemplo no serían adecuadas para escribir un sistema de nómina. Pero hay algunas áreas en las que se han aplicado exitosamente como son: en el Análisis financiero, el Procesamiento de señales, en la Mercadotecnia, en la Automatización y robótica, en los Diagnósticos médicos, en problemas de Clasificación, en el Reconocimiento de patrones, en el Control de procesos, y sobre todo muy importante para este trabajo de tesis en la Optimización [GOG99].

4.6.4 Alcances y Limitaciones de las RNA

Nace entonces la inquietud de saber : ¿ las RNA, en qué tipo de aplicaciones trabajan adecuadamente y en dónde su aplicación es potencial?, y por ende ¿cuáles son los alcances y limitaciones de las RNA?, la respuesta es que trabajan exitosamente en tareas donde no hay reglas bien definidas las cuales parecen fáciles para los humanos y difíciles para las computadoras. Este campo es tan amplio como el de las computadoras convencionales. La Inteligencia Artificial ha estado generalmente dominada por las áreas de manipulación lógica y simbólica, pero algunos piensan que las RNA reemplazarán la Inteligencia Artificial tradicional actual. Mas bien parece que se combinarán en sistemas apoyándose mutuamente, como sucede en los seres humanos, pues éstos se apoyan en sistemas rápidos soportados por reconocimiento de patrones. Otros sistemas realizan actividades que requieren más tiempo cuando el reconocimiento falla o cuando se requieren niveles superiores de decisión [GOG99], [ZUR92].

4.6.5 Confiabilidad de las RNA

Otra pregunta interesante es ¿qué tan confiables son las RNA?, a lo que se puede responder que las RNA son hasta cierto grado impredecibles (como los humanos). No hay manera de garantizar la salida de una red para una entrada a menos que se prevean todas las posibilidades en las entradas, y se entrene la red suficientemente bien. Sin embargo esto puede resultar impráctico en muchos casos y verdaderamente imposible en otros, lo que ha originado muchas críticas a las RNA. Parte de esta crítica se debe al hecho de que esperamos que las computadoras sean perfectas, pero los humanos no son perfectos... Otro problema relacionado y criticado en las RNA es su inhabilidad de "explicar" como resuelven el problema. Esta inhabilidad se debe a que la representación interna generada en la red puede ser demasiado compleja aún en los casos mas sencillos [GOG99], [ZUR92].

4.6.6 El Neurón Artificial

El neurón artificial fue diseñado para simular las principales características de un neurón biológico [WAS89]. El proceso típico de un neurón artificial es sencillo pues consiste básicamente en recibir un conjunto de valores de entrada, cada una representando la salida de otro neurón, o una entrada del medio externo, realizar una suma ponderada con estos valores, y "filtrar" este valor con una función de activación, como se muestra en la figura 4.13 [GOG99]. Muchos autores representan a los neurones con un círculo, como muestra la figura 4.14, dando como entendido que dentro del círculo se lleva a cabo la suma ponderada con los valores de entrada. Las señales que llegan a la sinapsis de una neurona biológica son equivalentes a las entradas, estas entradas son llamadas x_1, x_2,….., x_n, y son almacenadas en un vector llamado **X** [WAS89]. Cada señal se multiplica por un peso que tiene asociado, $W_1, W_2 \ldots W_n$. Los pesos son almacenados en un vector llamado **W** [GOG99].

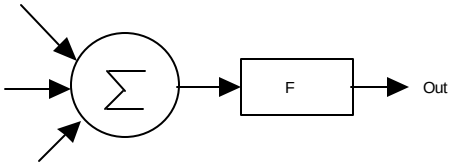


Figura 4.13 Cálculo de la salida de un neurón artificial

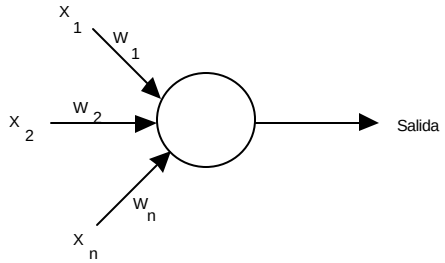


Figura 4.14 El Neurón Artificial

Cada peso corresponde a la "intensidad" o fuerza de la conexión de una sinapsis en un neurón biológico. Estas multiplicaciones se suman y este "sumador" es el que

corresponde vagamente al cuerpo de una neurona biológica . Como podemos observar, ésta es una suma algebraica de las entradas ponderadas que matemáticamente puede escribirse como [GOG99] :

$$NET = X_1W_1 + X_2W_2 + X_3W_3 + \ldots\ldots + X_nW_n$$

$$NET = \sum_{j=1}^{n} X_j W_j$$

Ó bien puede representarse en notación matricial como se expresa a continuación [GOG99] :

$$NET = \mathbf{XW}$$

Esta última ecuación puede representarse gráficamente como se muestra en la figura 4.15 [WAS89] :

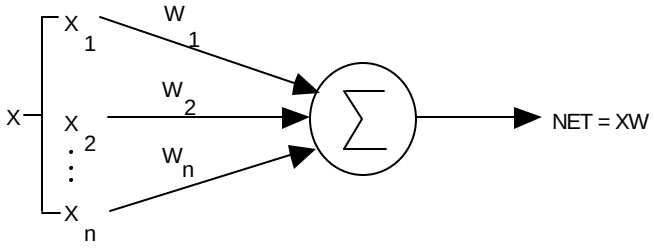


Figura 4.15 Neurón artificial matemático

4.6.6.1 Funciones de Activación

La señal NET generalmente se procesa por medio de una función de activación F, la cual producirá una señal que será la salida del neurón, ésta señal se llamará OUT [GOG99] :

$$OUT = F(NET)$$

Un ejemplo de una función de activación puede ser una función lineal:

$$OUT = K(NET) \qquad K = \text{Constante}$$

ó una función umbral como se muestra en la figura 4.16 :

$$OUT = \begin{cases} 1 & Si\_NET > T \\ 0 & En\_caso\_contrario \end{cases}$$

donde T es un valor de umbral

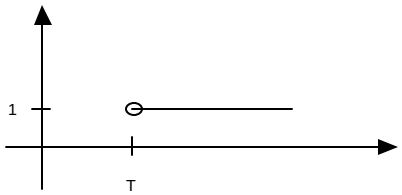


Figura 4.16 Función umbral

F también puede ser alguna otra función que simule mejor las características no lineales de transferencia de una neurona biológica. Si F reduce el rango de NET de manera que OUT nunca se salga de algún límite, independientemente de lo grande que sea NET, entonces F es una función "squash". Una función squash muy popular es la "sigmoide" o función logística como se muestra en la figura 4.17 [GOG99] :

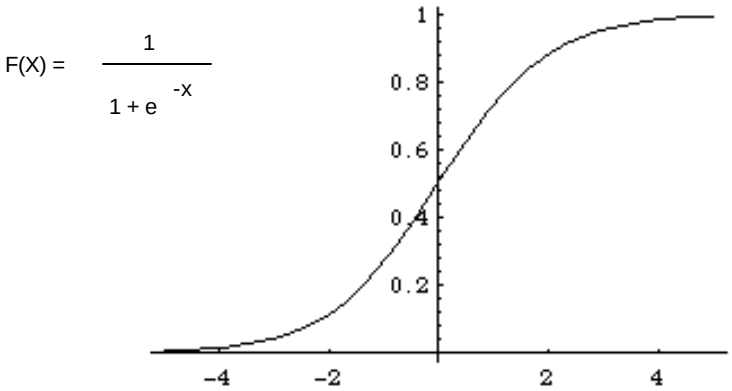


Figura 4.17 Función sigmoide ó logística

Utilizando la notación del neurón artificial, tendríamos que:

$$OUT = \frac{1}{(1 + e^{-NET})}$$

Podemos pensar que la función de activación determina la ganancia no lineal del neurón. Esta ganancia se puede definir como la razón de cambio en OUT con respecto a cambio en NET. Nótese que en los valores pequeños de NET el cambio de OUT es alto, mientras que para un valor de NET muy grande, el cambio en OUT es mínimo. Grossberg (1973) encontró que este tipo de funciones no lineales resuelven un problema de saturación y ruido muy común en las RNA [GOG99], [ZUR92].

Otra función muy común es la tangente hiperbólica que se muestra en la figura 4.18, esta es usada entre otras en la Red de Hopfield. La forma es similar a una Función sigmoide ó logística y es usada por biólogos como un modelo matemático de activación nervioso. Está dada por :

OUT = Tanh (NET)

La función Tanh se calcula con la siguiente ecuación [GRA92] :

$$Tanh(v) = \frac{Senh(v)}{Cosh(v)} = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

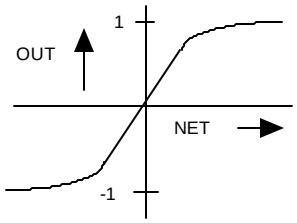


Figura 4.18 Función tangente hiperbólica

La función tangente hiperbólica tiene forma de "S" siendo simétrica en el origen, y a diferencia de la función sigmoide, tanh puede dar valores bipolares para OUT, es decir valores positivos y negativos, lo cual es útil en algunos modelos [WAS89]. Nótese que este modelo de neurón simplificado no considera el tiempo en ningún punto, sino que supone una salida instantánea. En otras palabras, no considera funciones de sincrónica [GOG99].

4.6.6.1 RNA de un nivel y RNA de varios niveles

Como podemos ver el modelo de un neurón artificial es sencillo; el poder de las RNA se obtiene de las conexiones entre neurones. Los neurones se relacionan entre sí formando redes que pueden llegar a ser tan complejas como el neocognitrón, o tan simples como el perceptrón. En general, se pueden identificar capas o niveles en una red. El modelo mas simple es el de una capa como se muestra en la fig. 4.19, en la misma figura se muestran los siguientes valores [GOG99] :

W = matriz de m renglones y n columnas
m = número de entradas
n = número de salidas
X = Vector renglón que contiene los valores de entrada

Sea **N** el vector de todos los valores NET y
N = **XW**
Y = F(**N**)
Y representa al vector de salida de la red. Por ejemplo en figura 4.19, con n = 3 **Y** se calcula de la siguiente forma:

Como se tiene Y(i) con i = 1, 2, …, n calcularíamos Y_1, Y_2, Y_3 porque n llega hasta 3

$Y_1 = F\ (x_1 w_{11} + x_2 w_{21} + x_3 w_{31})$
$Y_2 = F\ (x_1 w_{12} + x_2 w_{22} + x_3 w_{32})$
$Y_3 = F\ (x_1 w_{13} + x_2 w_{23} + x_3 w_{33})$

Estas ecuaciones pueden escribirse en notación matricial, de la siguiente forma :

$$Y = [X_1 X_2 X_3] \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix}$$

$$Y = [X_1 W_{11} + X_2 W_{21} + X_3 W_{31},\ X_1 W_{12} + X_2 W_{22} + X_3 W_{32},\ X_1 W_{13} + X_2 W_{23} + X_3 W_{33}]$$

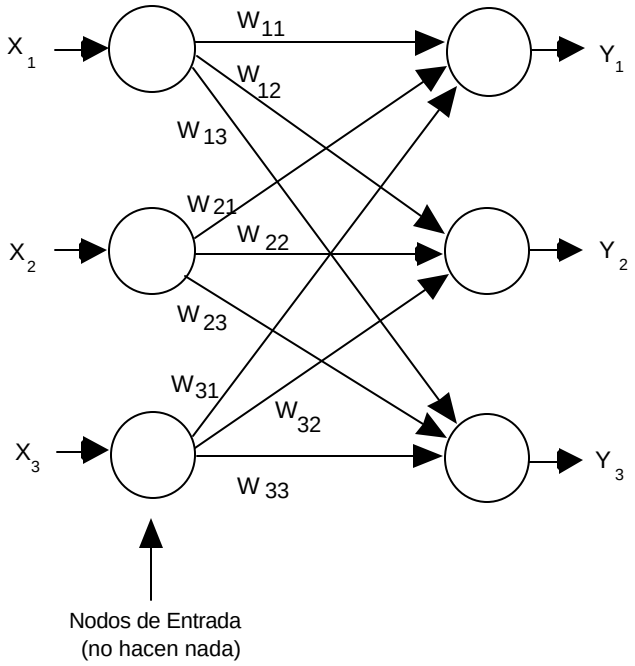


Figura 4.19 Red neuronal de 1 nivel

Es importante hacer notar que algunos autores no se ponen de acuerdo con respecto al número de capas que se dice tiene una red. Esto es porque algunos cuentan el nivel de entrada como una capa, y otros no. Wasserman no considera la capa de neurones de entrada como uno de los niveles de la red 4.20.b [GOG99], [WAS89] Se hace la aclaración que este será el concepto que se tomará en este documento. Por lo tanto el ejemplo de la figura 4.20.a muestra una red neuronal de 1 nivel escondido, y una red neuronal de 2 niveles escondidos en la figura 4.20b [GOG99], [WAS89].

En una red con varios niveles o capas, la función de activación debe ser no lineal, ya que de no ser así una red con varios niveles equivaldría a una red con un nivel.

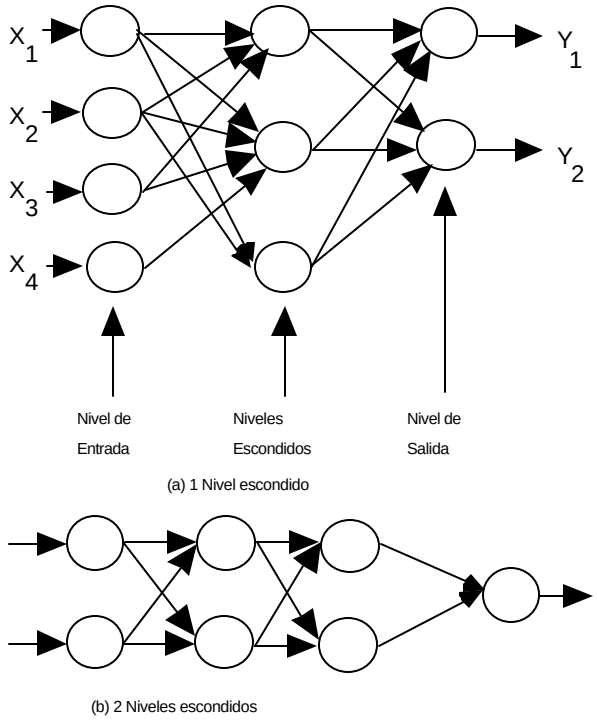


Figura 4.20 Ejemplos de RNA de varios niveles

Esta afirmación puede demostrarse si se considera que F es una función lineal, entonces la salida del primer nivel de la red esta dada por la siguiente ecuación [GOG99]:

$$\overline{OUT}_1 = F(\overline{X} \cdot \overline{W}_1)$$
$$\overline{OUT}_1 = K_1 \cdot \overline{X}\ \overline{W}_1 \qquad \text{Si F es lineal,}$$

y en la salida del 2do. nivel de la red neuronal se tiene :

$$\overline{OUT_2} = F(OUT_1 W_2) = F(K_1 \cdot \overline{X}\ \overline{W_1}\ W_2) = K^2 \overline{X}(W_1 W_2)$$

este resultado equivaldría a tener solo un nivel, ya que siempre se pueden hallar valores para una W_3 que sea [GOG99] :

$$W_3 = \overline{w}_1 \cdot \overline{w}_2$$

4.6.6.1 RNA no-recurrentes y recurrentes

En las redes no-recurrentes no hay retroalimentación de las salidas de un nodo hacia sus entradas. La no retro-alimentación asegura estabilidad, sin embargo, las redes no recurrentes están limitadas en comparación de las recurrentes. Inestabilidad significa que la red va de estado a estado infinitamente sin detenerse. Por el contrario en las redes recurrentes, se calcula su valor de salida, el cual es re-alimentado como entrada, recalculándose la salida otra vez. Los cambios en la salida supuestamente van siendo más y más pequeños hasta llegar a cero, en donde la red se "estabiliza", aunque puede ser que esto nunca suceda (ejemplo: un sistema caótico). Entre las redes recurrentes más

populares se encuentran las Redes de Hopfield. Una de las principales contribuciones de Hopfield es la conceptualización de las redes neuronales como sistemas dinámicos con energía [GOG99].

4.6.6.1.1 Configuración de la Red de Hopfield

La red de Hopfield memoriza patrones y es capaz de suministrar patrones correctos de la recepción de formas perturbadas de los mismos, ver la figura 4.21 como un ejemplo [GAG99] :

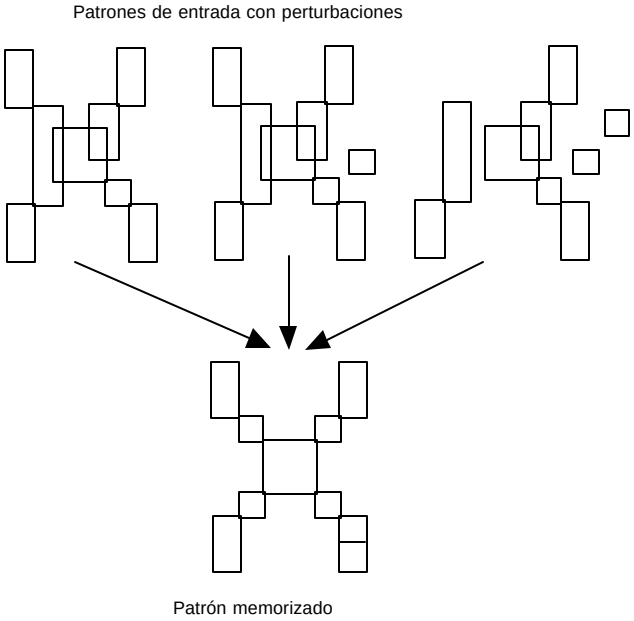


Figura 4.21 Patrones con perturbaciones

La red de Hopfield se puede utilizar como una memoria asociativa o para resolver problemas de optimización. Una *memoria asociativa* ó *memoria dirigida* por contenido es útil cuando se cuenta con parte de un patrón de entrada y se requiere el patrón completo. Como ejemplo de memoria asociativa citemos una referencia bibliográfica. Si contamos con una referencia bibliográfica incompleta podemos obtenerla completa a través de una memoria asociativa [GOG99].

El modelo básico de Hopfield es el siguiente [GOG99] :

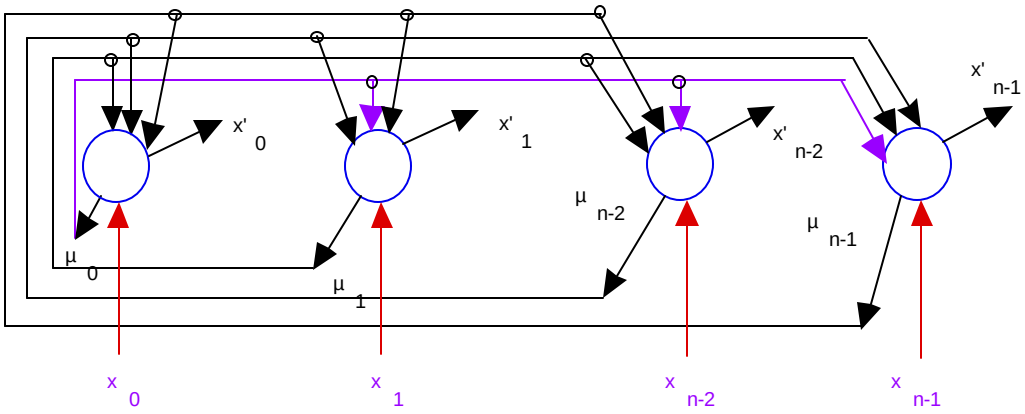


Figura 4.22 Red de Hopfield

El funcionamiento de la red es el siguiente: se aplica el vector X en el tiempo t =0. La red empieza a calcularse y recalcularse hasta que los valores μ´s ya no cambian. Cuando las salidas ya no cambian, se dice que es el momento de convergencia y entonces $X_o \ldots X_{n-1} = \mu_o' \ldots \mu_{n-1}'$ [GOG99].

Otras características de la red son [GOG99] :
- La función de activación es la escalón.
- Los patrones de entrada y salida pueden tomar valores +1 ó -1.
- La salida de cada nodo retroalimenta a los otros nodos vía pesos, que llamaremos t_{ij}.

4.6.6.1.2 Entrenamiento y Operación de la Red de Hopfield

Para la Red de Hopfield, el cálculo de los pesos (entrenamiento de la red) es muy sencillo, consiste simplemente en determinar el valor de la matriz de pesos T por medio de multiplicación de matrices. Una sinapsis entre dos neurones es definido por una conductancia T_{ij} la cual conecta una de las dos salidas del amplificador j a la entrada del amplificador i [HOT85]. Por otro lado, el proceso de evaluación de la red, o recuperación de información, es iterativo; y finaliza cuando la red converge [GOG99].

De manera general, una red de Hopfield funciona así [GOG99] :

Suponiendo M clases a almacenar, en una red de Hopfield con N nodos.

1) Se calculan los pesos T_{ij} de acuerdo a una receta dada, que involucran las M clases (ó patrones que se desean almacenar en la memoria).
2) Se muestra a la red un patrón cualquiera (completo o incompleto).
3) La red empieza a iterar en pasos discretos de acuerdo a una fórmula dada, hasta que la salida converge, es decir, no cambia más.
4) El patrón mostrado en este momento por la red es el patrón de salida.

Los pasos 2 a 4 se repiten para los patrones que se deseen recuperar de la red. Hopfield, Cohonen y Grossberg mostraron que esta red converge si los pesos son simétricos, esto es si [GOG99], [HOT85] :

$$t_{ij} = t_{ji} \text{ y } t_{ij} = 0 \text{ para todo i.}$$

4.6.6.1.2 Algoritmo de Hopfield

De manera más formal, el algoritmo de Hopfield es el siguiente [GOG99], [HOT85] :
Paso 1. Calcule los valores de los pesos que conectan los nodos utilizando la fórmula.

$$t_{ij} = \begin{cases} \sum_{s=0}^{M-1} X_i^s X_j^s & i \neq j \\ 0 & i = j \end{cases}$$

donde: t_{ij} es el peso que va del neurón i al j.
X_i^s es el valor del i-ésimo elemento (nodo) de la s-ésima clase.
M es el # de clases que se desean aprender.

En notación matricial tenemos que :

$$\bar{T} = \sum_{i=0}^{M-1} \bar{X}_i^T \bar{X}_i$$

Donde $\bar{X}_i$ es el i-ésimo patrón a almacenar.

Esto se conoce como el producto externo (Outer Product) de un vector-renglón consigo mismo .

Paso 2. Inicialize la red con el patrón de entrada dado (**X**), esto es

$$\mu_i(0) = X_i \qquad 0 \leq i \leq N\text{-}1$$

donde:

$\mu_i(0)$ es la salida de la red en el tiempo t = 0.

X_i puede ser +1 ó -1, y es el i-ésimo elemento del patrón de entrada. N es el número de entradas de la red.

Paso 3. Itere hasta converger de acuerdo a la fórmula:

$$m_j(t+1) = F_h\left[\sum_{i=0}^{N-1} t_{ij}\, m_i(t)\right] \qquad 0 \leq j \leq N\text{-}1$$

Donde:

$$F_h(x) = \begin{cases} 1 & \text{si } x > 0 \\ -1 & \text{si } x < 0 \\ m_j(t) & \text{si } X = 0 \text{ esto es, no hay cambio en el valor} \\ & m_i(t) \text{ en el vector } m. \end{cases}$$

La salida de la red estará dada después de la convergencia. Esta salida representa al patrón que más se parece al patrón de entrada dado.

4.6.6.1.3 Limitaciones de la Red de Hopfield

Las principales limitaciones de la red de Hopfield son [GOG99], [HOT85] :
1) El número de patrones que se pueden almacenar depende del número de nodos en la red, y si se tratan de almacenar muchos patrones, la red converge a patrones desconocidos. Hopfield mostró que esto ocurre raramente si el número de patrones almacenados (M) es menor que 0.15 N, donde N =#de nodos.
2) La red es inestable si hay poca ortogonalidad entre patrones, esto es, un patrón tiene varios bits en común con otro [GOG99].

4.6.6.1.4 La Red de Hopfield como solución al PAV

En 1985, Hopfield y D.W.Tank publicaron un artículo llamado "Neural Computation of Decisions in Optimization Problems", en el cual proponen una solución al problema del agente viajero (Traveling Salesman Problem,TSP). La solución encontrada por Hopfield no es la óptima, sin embargo las respuestas se encuentran mas o menos rápidamente [GOG99]. Algunos investigadores calculan que usando este método se puede encontrar una solución después de N^3 cálculos. La solución obtenida por este método depende de parámetros que son difíciles de asignar, y muchas veces obtiene resultados más bien malos, sin embargo es ingeniosa [GOG99].

4.6.6.1.5 Definición del PAV usando la Red Neuronal de Hopfield

Supóngase que hay N ciudades: A, B, C, D, etc. a visitar y que las distancias entre ellas está dada por el valor d_{xy} que es la distancia de la ciudad x a la ciudad y . Una ruta posible para el agente viajero se puede representar como un conjunto de N renglones (uno por cada ciudad) de 0´s y 1´s donde en cada renglón hay un solo 1, el cual indica la posición de esa ciudad en la ruta. Una ruta válida tiene sólo un uno por renglón y un uno por columna. Por ejemplo, suponiendo que el agente debe visitar 4 ciudades A, B, C y D podemos definir a una posible "ruta" con la siguiente matriz [GOG99]:

		Orden	de	Visita
Ciudad	1	2	3	4
A	0	1	0	0
B	0	0	0	1
C	1	0	0	0
D	0	0	1	0

Tabla 4.1 Matriz que representa la ruta C-A-D-B

Se localiza la columna 1 de orden de visita y sobre esa columna se busca la dirección de la fila donde se encuentra el 1, esta fila corresponde a la ciudad No. 1, en este caso C. Se realiza la misma operación ahora con la columna 2, 3, etc. obteniendo así la ruta C-A-D-B. Esta matriz se puede representar en un vector de una dimensión con N^2 elementos. Dicho vector puede representarse en una red de Hopfield con N^2 neurones [GOG99].

El objetivo de la red de Hopfield es hacer converger la red hacia una ruta válida en la cual exista la mínima energía posible. El problema está entonces en definir ahora que es la "energía" del sistema [GOG99].

4.6.6.1.5 Función de Energía

La forma de definir la función de energía no es única, pero requiere de las especificaciones del problema y que esta sea minimizada [ARP97]. Por lo que se puede notar que la representación escogida del problema establece la restricción que hay un solo uno por renglón y un sólo uno por columna, Además, sabemos que el objetivo es minimizar la distancia entre ciudades [GOG99].

Hopfield y Tank definieron la siguiente fórmula de energía que contiene dichas restricciones :

$$E = \frac{A}{2}\sum_{x}\sum_{i}\sum_{j\neq i} V_{x,i}\,V_{x,j} + \frac{B}{2}\sum_{i}\sum_{x}\sum_{y\neq x} V_{x,i}\,V_{y,i} + \frac{C}{2}\left(\sum_{x}\sum_{i} V_{x,i} - n\right)^2 + \frac{D}{2}\sum_{x}\sum_{y\neq x}\sum_{i} d_{x,y}\,V_{x,i}\left(V_{y,i+1} + V_{y,i-1}\right)$$

Donde $V_{x,y}$ representa la salida del neurón que muestra que la ciudad x está en la posición y.

En la función se puede notar lo siguiente:

- el primer término es cero si y sólo si hay un solo "uno" en cada renglón. De otra manera, el término toma en valor mayor que cero.
- el 2º término es cero si y sólo si hay sólo un uno en cada columna.
- el 3º término será cero si y sólo si hay exactamente N 1´s en la matriz ruta.
- el 4º término representa la longitud de una ruta válida.

4.6.6.1.6 Cálculo de la Red

La salida de cada nodo se calcula de la siguiente forma [GOG99] :

$$V_{x,i} = g(\mu_{x,i}) = 1/2(1+\mathrm{Tanh}(\mu_{x,i}/(\mu_o))$$

Donde $\mu_{x,i}$ representa la entrada de cada neurón. Su valor deberá cambiar de manera que se reduzca la energía del sistema, este cambio se puede escribir como:

$$m_{x,i} = m_{x,i} + \left(\frac{dm_{x,i}}{dt}\right)\frac{1}{\Delta t}$$

Donde :

$$\frac{dm_{x,i}}{dt} = -\frac{m_{x,i}}{t} - A\sum_{j\neq i} V_{x,j} - B\sum_{y\neq x} V_{y,i} - C\left(\sum_{x}\sum_{j} V_{x,j} - n\right)^2 -$$

$$D\left(\sum_{y} d_{x,y}\ \left(V_{y,i+1} + V_{y,i-1}\right)\right)$$

4.6.6.1.7 Valores de los parámetros

A, B, C, D, μ_o, Δt y μ deben tomar un valor inicial.
Desgraciadamente la convergencia de la red va a depender de estos valores, los cuales son realmente difíciles de establecer. Hopfield y Tank han sugerido que el valor inicial sea : μ_o = 1/N añadiendo un poco de ruido para romper la simetría, esto es:

$\mu_o = \mu_o$ + noise(0.1* μ inicial)
Donde
noise(x) = # al azar entre 0 y x
En algunas pruebas que realizaron Hopfield y Tank utilizaron los siguientes valores:
A = 1000
B = 1000
D = 1000
C = 0.75*D*N
μo = 0.02
τ = 1
•t = 20000, o bién 1/•t = increment = 0.00005

Hopfield y Tank mostraron sus resultados en un experimento con 10 ciudades. De 20 corridas que hicieron, 16 dieron rutas válidas y cerca del 50 % de las 50 soluciones fueron de las más cortas halladas por el método exhaustivo [Hopfield 87].

4.6.6.1.7 Algoritmo de la Red de Hopfield

Este algoritmo se tomó del Software realizado en "C" por la Doctora Pilar Gómez Gil, en Texas Tech University el mes de abril de 1990. Este se muestra en la figura 4.23.

```
{01} Procedure Red_Neuronal_Hopfield
{02} for x ← 0 to n - 1
{03}    begin
{04}       for i ← 0 to n - 1
{05}          begin
{06}             lownoise ← -0.1 * u0;
{07}             upnoise   ← 0.1  * u0;
{08}             u [x][i] ← u00 + un punto aleatorio en el intervalo(lownoise,upnoise);
{09}             v [x][i] ← (1/2)*[1+tanh(u(x,i)/u0)]
{10}          end del for i
{11}    end del for x
```

{12} do
{13} begin
{14} for x ← 0 to n - 1
{15} begin
{16} for i ← 0 to n - 1
{17} begin
{18} resta ←

$$-A\sum_{j\neq i} V_{x,j} - B\sum_{y\neq x} V_{y,i} - C\left(\sum_{x}\sum_{j} V_{x,j} - n\right)^2 - D\left(\sum_{y} d_{x,y}(V_{y,i+1} + V_{y,i-1})\right)$$

{19} u [x][i] ← u[x][i] + (- u [x][i] + resta) * increment;
{20} v [x][i] ← (1/2) [1+tanh(u(x,i)/u0)]
{21} end del for i
{22} end del for x
{23} st ← $\sum_{x=0}^{n-1}\sum_{i=0}^{n-1} V(x,i)$;
{24} past_energy ← energy;
{26} energy ←

$$\frac{A}{2}\sum_{x}\sum_{i}\sum_{j\neq i} V_{x,i}\,V_{x,j} + \frac{B}{2}\sum_{i}\sum_{x}\sum_{y\neq x} V_{x,i}\,V_{y,i} + \frac{C}{2}\left(\sum_{x}\sum_{i} V_{x,i} - n\right)^2 +$$

$$\frac{D}{2}\sum_{x}\sum_{y\neq x}\sum_{i} d_{x,y}V_{x,i}\,(V_{y,i+1} + V_{y,i-1});$$

{26} ite ← ite +1;
{27} imprime ("ite = ",ite," energía = ",energy);
{28} imprime ("Sumatoria de Vxi = ",st);
{29} dif = st - n_vertices;
{30} end del do-while;
{31} while ((abs(dif) > epsilon) & (ite<= maxite));

{32} i ← find_max(0,v);
{33} ant_city ← i;
{34} ini_city ← i;
{35} imprime ("Solución");
{36} imprime ("Visitar las ciudades en el siguiente orden");
{37} for j ← 0 to n
{38} begin
{39} i ← find_max(j,v);
{40} imprime ("La ciudad # = ",I," está en la posición = ",j);
{41} tot_distance ← tot_distance + dis[ant_city][i];
{42} ant_city ← i;
{43} end del for j
{44} tot_distance ← tot_distance + dis[ant_city][ini_city];
{45} imprime ("La distancia total es w = ", tot_distance);
{46} imprime ("El último valor de energía = ", energy);
{47} imprime ("El número total de iteraciones = ", ite);
{48} end método Red_Neuronal_Hopfield

Figura 4.23 Algoritmo de la Red Neuronal de Hopfield

Se explicarán las líneas más importantes

{02} a {11} Inicializa la matriz v [x][i], donde n es el número de vértices del PAV.
{12} a{31} Se repite hasta que la red converja.
{18} Se mejora u [x][i] en cada iteración que es la entrada de cada neurón. Su valor deberá cambiar de manera que se reduzca la energía del sistema.
{19} 1/•t = increment default = 0.0001
{20} Se mejora v [x][i] en cada iteración que es la salida de cada nodo o neurón.
{23} Se calcula la energía con la nueva v [x][i]
{01} a {31} Se calcula la red de Hopfield.
{32} a {48} Se despliega la solución.

4.6.6.1.8 Mejora a la Salida de la Red Neuronal de Hopfield

La propuesta de solución de Hopfield y Tank para el PAV es de un interés particular por su ingeniosa implementación. En el artículo [HOT85] fue reportada la solución de hasta 30 ciudades, sin embargo Wilson y Pawley reestudiaron esta red y llegaron a la conclusión de que para una ruta aceptable que cumpla con todas las restricciones, esto es difícil de alcanzar y que su posibilidad decrece rápidamente al aumentar el número de ciudades, de hecho para un problema PAV de 10 ciudades, después de 1000 iteraciones de la red una ruta aceptable se obtuvo en solo un 8% de todos los casos [ARP97]. Ellos también trataron de mejorar el método de solución mejorando el algoritmo en tres formas : dándole valores excesivamente grandes a un spin variable[2] de configuraciones imposible como V_{ii}, variando los valores de los parámetros, y cambiando las condiciones iniciales [HOT85]. Estas dos últimas variaciones son las que se implementaron en el software, pero a pesar de estos cambios el resultado final no mejoró mas que los resultados originales. Sin embargo con un nuevo truco presentado en este proyecto se le dio una nueva luz a este problema. Los resultados de la red tenían el problema de que en ocasiones daban rutas no válidas, es decir producía rutas con nodos repetidos. Este punto es una restricción al PAV que debe recorrer todos los nodos sin repetir dos veces el mismo. La solución consiste entonces en verificar si hay nodos repetidos. Si hay nodos repetidos, entonces se inserta en la posición del segundo nodo repetido uno no existente dentro de la ruta, de esta manera el resultado será siempre una ruta válida.

[2] Spin variable es el estado del neurón V(i,j), que significa que la ciudad i es visitada en el orden j.