COMPARACIÓN DE MÉTODOS DE REDES NEURONALES Y PROGRAMACION LINEAL ENTERA, PARA LA OPTIMIZACIÓN DE LA TRAYECTORIA DE MECANIZADO DE CAVIDADES COMPLEJAS

ERICK FERLEY APONTE BARAJAS
BRANDONT ALEXIS BAUTISTA ACOSTA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICOMECANICAS
ESCUELA DE INGENIERIA MECANICA
BUCARAMANGA
2017

COMPARACIÓN DE MÉTODOS DE REDES NEURONALES Y PROGRAMACION LINEAL ENTERA, PARA LA OPTIMIZACIÓN DE LA TRAYECTORIA DE MECANIZADO DE CAVIDADES COMPLEJAS

ERICK FERLEY APONTE BARAJAS
BRANDONT ALEXIS BAUTISTA ACOSTA

Trabajo de Grado para optar al título de
Ingeniero Mecánico

Director
ISNARDO GONZALEZ JAIMES
Ingeniero Mecánico

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICOMECANICAS
ESCUELA DE INGENIERIA MECANICA
BUCARAMANGA
2017

A el gran Dios y salvador Jesucristo, quien da sentido a mi vida, todo lo que soy y tengo se lo debo a Él, por ser mi inspiración, motivación y darme el privilegio de escalar hasta este peldaño de mi vida.

A mis padres Norberto y Georgina, por su amor incondicional; mi gran motor, su apoyo, consejos, paciencia y confianza.

A mis hermanos, Ingrid por brindarme su mano y compañía en este proceso de formación profesional, Melisa por sus ánimos y Jovany por la motivación que me brindó.

A Jessica, por su amor, apoyo incondicional y por cada palabra de aliento que me imprimía paciencia y motivación.

A mis hermanos y amigos de la Iglesia Pentecostal Unida de Colombia, por sus concejos, oraciones; y cada voz de aliento fue muy importante en mi desarrollo profesional y espiritual.

BRANDONT.

Primero que todo darle gracias a Dios por hacer posible llegar a este punto de mi vida

A mis padres Carmen Cecilia y William por su apoyo

A mis abuelos, especialmente a mi abuela Imelda Hernández, por darnos su colaboración en los momentos de mayor necesidad

Por ultimo a toda mi familia y amigos que de una u otra manera estuvieron aportando su granito de arena.

ERICK

AGRADECIMIENTOS

Los autores manifiestan sus agradecimientos:
A la Universidad Industrial de Santander y a la Escuela de Ingeniería Mecánica por la formación dada como profesionales en esta rama de la ciencia.
A Isnardo González Jaimes, Ingeniero Mecánico, director del proyecto y amigo, por su respaldo, confianza y apoyo sincero.
A todos los compañeros que de una u otra forma contribuyeron y enriquecieron nuestra formación integral.
A todos nuestros familiares, que nos motivaron a culminar y a no desfallecer durante este gran trabajo.

RESUMEN

TITULO:
COMPARACIÓN DE MÉTODOS DE REDES NEURONALES Y PROGRAMACION LINEAL ENTERA, PARA LA OPTIMIZACIÓN DE LA TRAYECTORIA DE MECANIZADO DE CAVIDADES COMPLEJAS.*

AUTORES:
ERICK FERLEY APONTE BARAJAS
BRANDONT ALEXIS BAUTISTA ACOSTA.**

PALABRAS CLAVES:
FRESADO DE CAVIDADES, DESBASTE, RED NEURONAL, MAPAS AUTO-ORGANIZADOS, CAD/CAM, PROBLEMA DEL AGENTE VIAJERO, PROGRAMACIÓN LINEAL ENTERA.

DESCRIPCION:
La finalidad de este proyecto es presentar una nueva perspectiva para la planeación de la trayectoria de la herramienta, realizando la comparación de métodos de redes neuronales y la programación lineal entera, orientados a la optimización de la trayectoria de mecanizado en el fresado, determinando el método más eficiente al momento de generar la trayectoria, y así dejar la base para la aplicación de dicho método a un software que optimice los procesos actuales de mecanizado en la industria metalmecánica; particularmente los métodos están compuestos de digitalización del área de la cavidad en un número finito de puntos, y el método a implementar para encontrar la trayectoria.
De todo este proceso se elabora un software de fácil manejo que presenta las diferentes trayectorias de la herramienta (verificándose longitud y tiempo de la herramienta) obtenida para cualquier cavidad formada arbitrariamente con numerosas islas cuyos resultados se presentan en un formato claro y de interpretación sencilla, que incluye la representación gráfica de la trayectoria de la herramienta y longitud de la misma permitiendo guardarlas en cualquier formato de imagen, con independencia del mismo software con el fin de establecer un patrón de comparación que permita determinar la viabilidad de cada método de obtención de la trayectoria.

* Proyecto de grado.

** Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería Mecánica, Ing. Isnardo González J.

SUMMARY

TITLE:
COMPARISON OF NEURAL NETWORKS METHODS AND INTEGER LINEAR PROGRAMMING, FOR THE OPTIMIZATION OF THE COMPLEX POCKET MILLING TRAJECTORY.*

AUTHORS:
ERICK FERLEY APONTE BARAJAS
BRANDONT ALEXIS BAUTISTA ACOSTA.**

KEY WORDS:
POCKET MILLING, ROUGHING, NEURAL NETWORK, SELF-ORGANIZING MAPS, CAD/CAM, TRAVELLING SALESMAN PROBLEM (TSP), INTEGER LINEAR PROGRAMMING (ILP)

DESCRIPTION:
The purpose of this project is to present a new perspective for the planning of the trajectory of the tool, making the comparison of neural network methods and the entire linear programming, oriented to the optimization of the machining path in the milling, determining the method more efficient at the moment of generating the trajectory, and thus leave the base for the application of this method to a software that optimizes the current processes of machining in the metalworking industry; particularly the methods are composed of digitizing the cavity area into a finite number of points, and the method to be implemented to find the path.
From all this process, an easy-to-use software is developed that presents the different trajectories of the tool (verified length and time of the tool) obtained for any cavity formed arbitrarily with numerous islands whose results are presented in a clear and simple interpretation format, which includes the graphical representation of the tool path and the length of the tool allowing to save them in any image format, independently of the same software in order to establish a comparison pattern that allows determining the viability of each method of obtaining the trajectory.

* Degree Project.

** Physical-mechanical Engineeries Faculty, Mechanical Engineering, Eng. Isnardo González J.

CONTENIDO

pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

pág.

LISTA DE ANEXOS

pág.

INTRODUCCION

Actualmente a nivel industrial las maquinas herramientas, son sin duda alguna, pieza fundamental en el desarrollo de una economía, por tanto cualquier avance o aporte tecnológico en aras de mejorar su eficiencia, genera un impacto positivo que se ve reflejado en el aumento de la productividad lo cual representa una mayor rentabilidad, de manera paralela es necesario la apropiación de esta tecnología y aplicarla a nuestro medio.

En la era moderna no basta solamente con concebir una idea de fabricar una maquina o elemento mecánico y materializarlos, sino que para llevar a cabo su realización se deben tener cuenta factores que algunos años atrás eran triviales (materiales, tiempo, impacto ambiental, etc.) debido a la revolución industrial esta perspectiva cambio, cambiando así la visión de diseño, la cual se hizo más genérica debido a que de alguna u otra manera nos repercute nuestro diario vivir

El mecanizado de cavidades complejas es una de las mayores operaciones de remoción de material en Control Numérico Computarizado (CNC), es hablar de un sin número de formas que varían desde una simple geometría a formas complejas, incluyendo islas, que son típicamente maquinadas en tres etapas: desbaste, corte intermedio y acabado. Debido a su complejidad requieren métodos especializados para generar las trayectorias de fabricación, se tienen diversos métodos que brindan una buena solución, sin embargo es de gran importancia generar un patrón de comparación con el fin de establecer que método es más eficiente, analizando como parámetro principal el tiempo y su fácil implementación.

En la presente investigación se hace referencia a los diferentes tiempos involucrados en la fabricación de piezas (tiempos de pasada, ocupación de máquinas, fabricación, etc.), cuyos resultados vienen a ser decisivos para el fresado de cavidades complejas, adicionalmente a presentar los conceptos de máquinas de

fresado, sus tipos, características principales y parámetros esenciales. En el fresado de cavidades complejas uno de los mayores inconvenientes en la optimización de condiciones de mecanizado es el de encontrar una trayectoria adecuada para la remoción de material, por esta razón hacemos un mayor énfasis en la optimización del tiempo principal (tp) aplicando la tecnología de redes neuronales y programación lineal entera con el fin de solucionar el problema de la trayectoria de la herramienta cuyo objeto es establecer que método se encamina como el mejor.

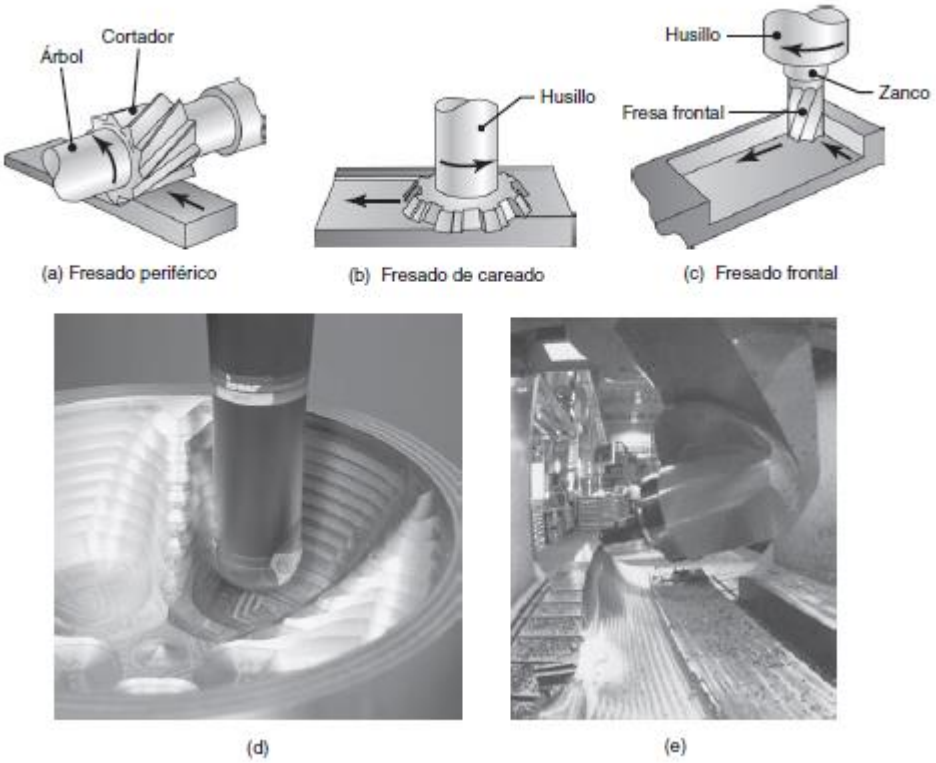
En la actualidad se ha desatado un auge en la investigación y desarrollo de las redes neuronales con el fin de implementarlas en cualquier campo de la ciencia, de igual manera el desarrollo de la programación lineal entera que se perfila como un método alterno a dar solución a problemas de optimización. Llama la atención que en nuestro país se puede observar como muchos programas educativos en universidades están incluyendo dentro de sus planes de estudio la tecnología de redes neuronales artificiales y la programación lineal entera. Este apogeo se debe principalmente a los buenos resultados que presentan en campos donde las técnicas convencionales o tradicionales no son aplicables o inviable y cuyo funcionamiento no ofrece resultados satisfactorios, por tanto es de vital importancia desarrollar métodos alternos.

En el siglo XXI donde se requiere cada vez más acelerar y optimizar todo, es muy importante el desarrollo de nuevas tecnologías, el presente trabajo ofrece una gran alternativa en el campo de la industria metalmecánica.

1. FRESADO

El fresado[1] incluye diversas operaciones de maquinado muy versátiles que tienen lugar en varias configuraciones (figura 1) usando una fresa, una herramienta multifilo que produce numerosas virutas en una sola revolución.

Figura 1. Algunos tipos básicos de cortadores y operaciones de fresado.



(a) Fresado periférico. *(b)* Fresado de careado o refrentado. *(c)* Fresado frontal. *(d)* Fresa de punta esférica con insertos indexables con cubierta de carburo maquinando una cavidad en una matriz. *(e)* Fresado de una superficie esculpida con una fresa frontal, utilizando una máquina de control numérico de cinco ejes

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.725. ISBN 978-970-26-1026-7

[1] KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. ISBN 978-970-26-1026-7

1.1 FRESADO PERIFÉRICO

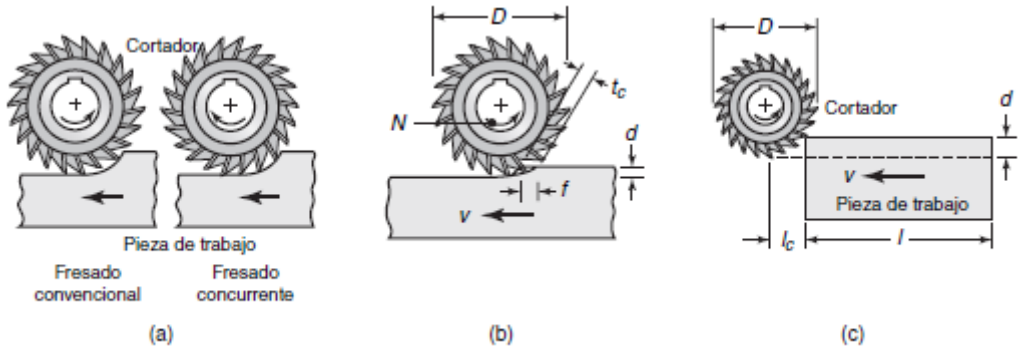
En este proceso (también conocido como fresado simple), el eje de rotación del cortador es paralelo a la superficie de trabajo, como se muestra en la Figura 1a. El cuerpo del cortador, que por lo general se fabrica con acero de alta velocidad, tiene varios dientes a lo largo de su circunferencia; cada diente actúa como una herramienta de corte de una sola punta. Cuando el cortador es más largo que la anchura del corte, el proceso se conoce como fresado plano. Los cortadores para fresado periférico pueden tener dientes rectos o helicoidales (Figura 1a), que producen una acción ortogonal u oblicua, respectivamente.

Por lo común, son preferibles los dientes helicoidales a los rectos porque el diente ataca en forma parcial la pieza de trabajo conforme va girando. En consecuencia, la fuerza de corte y el torque en el cortador son inferiores, lo que ocasiona una operación más suave y reduce el traqueteo.

1.1.1 Fresado convencional y fresado recurrente. En la figura 2a se observa que el cortador puede rotar en el sentido de las manecillas del reloj o en sentido opuesto; esto es significativo en la operación. En el fresado convencional (también conocido como fresado hacia arriba), el máximo espesor de las virutas se da al final del corte, cuando el diente abandona la superficie de la pieza de trabajo. Las ventajas del fresado convencional son que (a) el ataque del diente no es una función de las características de la superficie de la pieza de trabajo, y (b) la contaminación o cascarilla (capa de óxido) en la superficie no afecta de manera adversa la vida de la herramienta. Éste es el método más común de fresado. El proceso de corte es fino, siempre que los dientes del cortador estén bien afilados. De lo contrario, el diente roza contra la superficie antes de empezar a cortar. También puede haber una tendencia a que la herramienta vibre y la pieza de trabajo se jale hacia arriba (debido a la dirección de la rotación del cortador), por lo que es necesaria una sujeción apropiada. En el fresado concurrente (también conocido como fresado hacia abajo), el corte empieza en la superficie de la pieza de trabajo donde la viruta

es más gruesa. La ventaja es que el componente hacia abajo de la fuerza de corte mantiene la pieza en su lugar, en particular las partes más delgadas. Sin embargo, dadas las fuerzas de impacto que se producen cuando los dientes atacan la pieza, la operación debe mantener rígida la sujeción del trabajo y eliminar el retroceso del engrane en el mecanismo de avance de la mesa. El fresado concurrente no es apropiado para el maquinado de piezas de trabajo que tienen cascarilla en la superficie, como los metales trabajados en caliente, los forjados y las fundiciones La cascarilla es dura y abrasiva y produce desgaste excesivo, así como daños a los dientes del cortador, lo que acorta la vida de la herramienta.

Figura 2. Tipos de fresado y parámetros



(a) Esquema del fresado convencional y del fresado concurrente. *(b)* Operación de fresado plano que muestra la profundidad de corte, *d*; avance por diente, *f*; profundidad de corte de la viruta, *tc*, y velocidad de la pieza de trabajo, *v*. *(c)* Esquema de la distancia de desplazamiento del cortador, *lc*, para alcanzar la profundidad de corte total.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.726. ISBN 978-970-26-1026-7

1.1.2 Parámetros del fresado. La velocidad de corte (V) en el fresado periférico es la velocidad superficial del cortador, o donde D es el diámetro del cortador y N la velocidad rotacional del mismo (Figura 3).

$$V = \pi DN$$

Obsérvese que el espesor de la viruta en el fresado plano varía a lo largo de su longitud debido al movimiento longitudinal relativo entre el cortador y la pieza de trabajo

Para un cortador de dientes rectos, el espesor no deformado de la viruta (profundidad de corte de la viruta) aproximado (t_c) se puede calcular a partir de la ecuación

$$t_c = 2f\sqrt{\frac{d}{D}}$$

donde f es el avance por diente del cortador, es decir, la distancia que avanza la pieza de trabajo por diente del cortador, en mm/diente o pulgadas/diente, y d es la profundidad de corte. Al aumentar t_c, se incrementa la fuerza en el diente del cortador.

El avance por diente se determina con la ecuación:

$$f = \frac{v}{Nn}$$

donde v es la velocidad lineal (velocidad de avance) de la pieza de trabajo y n la cantidad de dientes en la periferia del cortador. La precisión dimensional de esta ecuación se puede verificar utilizando las unidades adecuadas para los términos individuales; así, por ejemplo,

(mm/diente)=(m/min)(10^3mm/m)/(rev/min)(número de dientes/rev).

El tiempo de corte (t) se obtiene mediante la expresión

$$t = \frac{l + l_c}{v}$$

Donde l es la longitud de la pieza de trabajo (Figura 2c) y l_c la extensión horizontal del primer contacto del cortador con la pieza de trabajo. Con base en la suposición de que $l_c << l$ (aunque por lo general éste no es el caso), la velocidad de remoción de material (MRR) es

$$MRR = \frac{lwd}{t} = wdv$$

en la que *w* es la anchura de corte que (en el fresado plano) es la misma que la anchura de la pieza de trabajo. La distancia que el cortador avanza en el ciclo sin corte de la operación de fresado es una consideración económica importante y debe minimizarse, por medios tales como un avance más rápido de los componentes de la máquina herramienta.

Resumen de parámetros y fórmulas del fresado periférico

N = Velocidad rotacional del cortador de fresado, [rpm]

F = Avance, [mm/diente] o [pulgadas/diente]

D = Diámetro del cortador, [mm] o [pulgadas]

n = Número de dientes del cortador

v = Velocidad lineal de la pieza de trabajo o velocidad de avance, [mm/min] o [pulgadas/min]

V = Velocidad superficial del cortador, [m/min] o [pies/min] = DN

f = Avance por diente, [mm/diente] o [pulgadas/diente] = $\frac{v}{Nn}$

l= Longitud de corte, [mm] o [pulgadas]

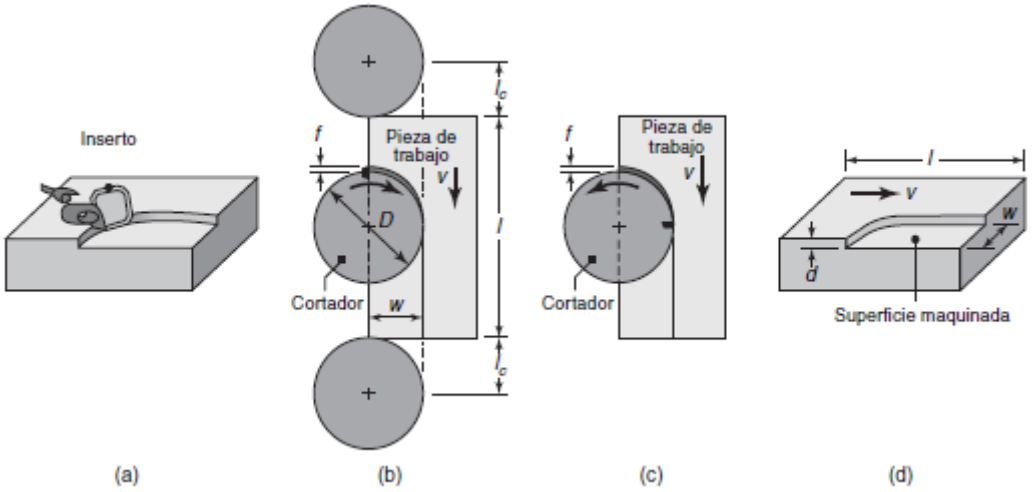
t = tiempo de corte, seg o min = $\frac{l+l_c}{v}$, en la que l_c = medida del primer contacto del cortador con la pieza de trabajo

MRR = [mm^3 /min] o [pulg^3 /min] = wdv, en la que *w* es el ancho del corte

Torque = [N·m] o [lb·pie] = *FcD*/2

Potencia = [kW] o [hp] = (Torque)(*w*), en la que *w* = 2πN [radianes/min]

Figura 3. Operación de fresado de careado



Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.726. ISBN 978-970-26-1026-7

1.2 FRESADO DE CAREADO O REFRENTADO

En el fresado de careado o refrentado (Figura 3), que muestra *(a)* la acción de un inserto en el fresado de careado; *(b)* fresado concurrente; *(c)* fresado convencional; *(d)* dimensiones en el fresado de careado. La anchura de corte, *w*, no es necesariamente la misma que el radio del cortador. El cortador se monta en un husillo que tiene un eje de rotación perpendicular a la superficie de la pieza de trabajo (Figura 1b) y remueve material en la forma mostrada en la figura 3a. La cortadora gira a una velocidad rotacional *N* y la pieza se mueve a lo largo de una trayectoria recta a una velocidad lineal *v*. Cuando la rotación del cortador es semejante a la de la figura 3b, la operación es *fresado* concurrente; cuando lo hace en la dirección opuesta (Figura 3c), la operación es fresado convencional. Los dientes de corte, como los insertos de carburo, se montan en el cuerpo del cortador como se muestra en la figura 4.

Dado el movimiento relativo entre los dientes del cortador y la pieza de trabajo, el fresado de careado deja marcas de avance en la superficie maquinada (Figura 5).

Obsérvese que la rugosidad de la superficie de la pieza depende de la geometría de la esquina del inserto y del avance por diente.

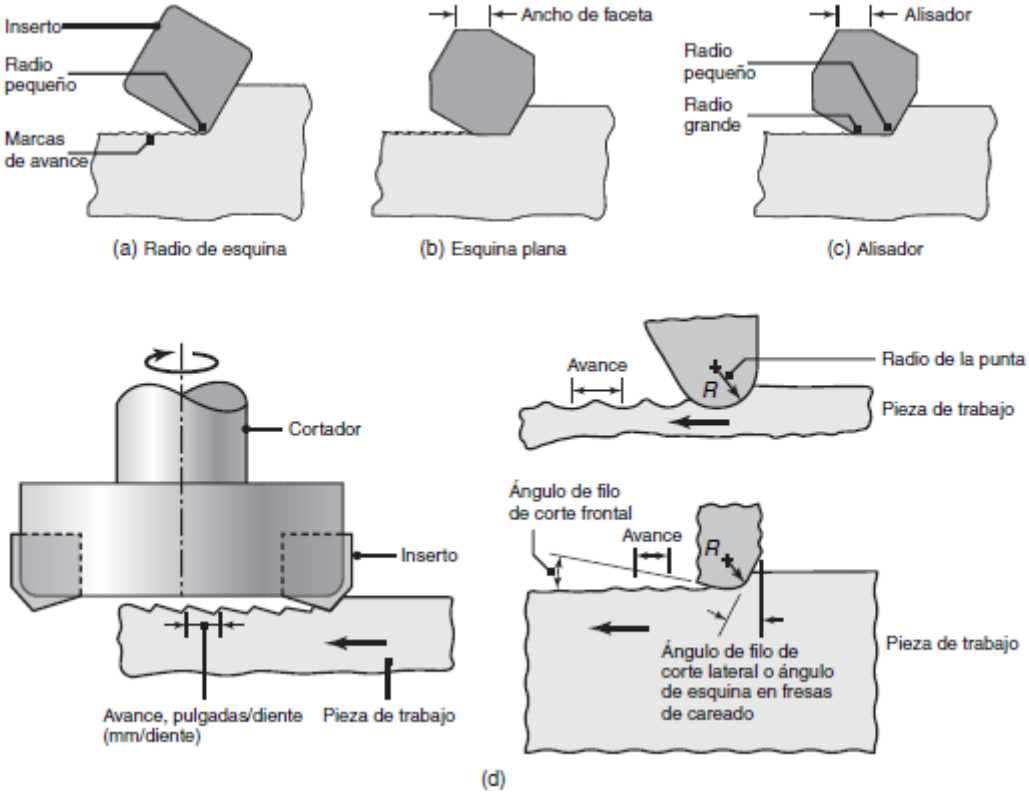
En la figura 6 se muestra la terminología de un cortador de fresado de careado o refrentado, así como los diversos ángulos. Como se puede ver en la vista lateral del inserto en la figura 7, el ángulo de avance del inserto en el fresado de careado influye de manera directa sobre el espesor no deformado de la viruta, como sucede en las operaciones de torneado. Al aumentar el ángulo de avance (positivo, como se muestra en la figura 7), disminuye el espesor no deformado de la viruta (como sucede con el espesor de la viruta) y se incrementa la longitud de contacto (anchura de la viruta). El ángulo de avance también influye en las fuerzas de fresado: es evidente que al disminuir dicho ángulo, se presenta un componente de fuerza vertical cada vez más pequeño (fuerza axial sobre el husillo del cortador). Por lo general, los ángulos principales para la mayoría de los cortadores de fresado de careado van de 0° a 45°. En la figura 7 se observa que el área transversal de la viruta sin deformar permanece constante.

Existe una amplia variedad de cortadores para fresado. El diámetro del cortador debe elegirse de manera que no interfiera en los aditamentos y otros componentes en el arreglo. En una operación común de fresado de careado, la relación del diámetro del cortador (D) a la anchura del corte (w) no debe ser menor a 3:2.

Figura 4. Cortador de fresado de careado con insertos indexables.

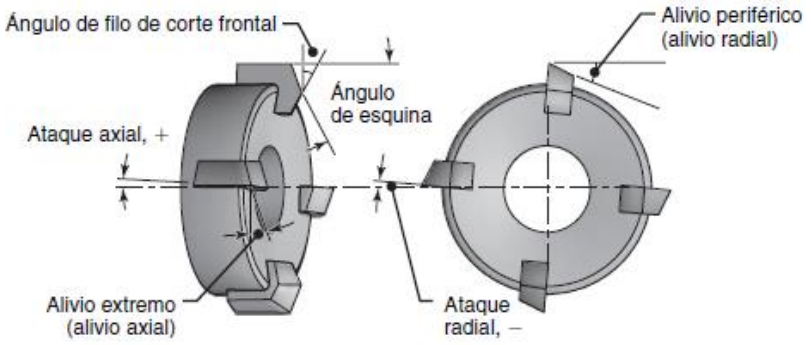
Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.729. ISBN 978-970-26-1026-7

Figura 5. Esquema del efecto de la forma del inserto sobre las marcas de avance en una superficie de fresado de careado



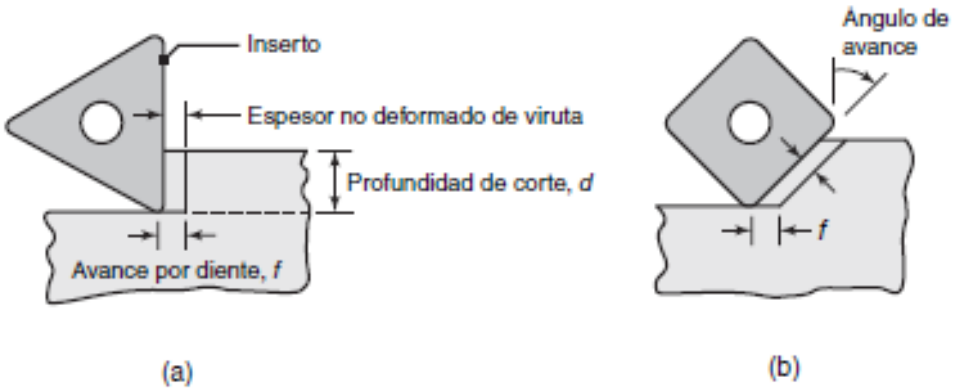
Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.730. ISBN 978-970-26-1026-7.

Figura 6. Terminología del cortador de fresado de careado.



Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.730. ISBN 978-970-26-1026-7

Figura 7. Efecto del ángulo de avance



Sobre el espesor no deformado de la viruta en el fresado de careado.

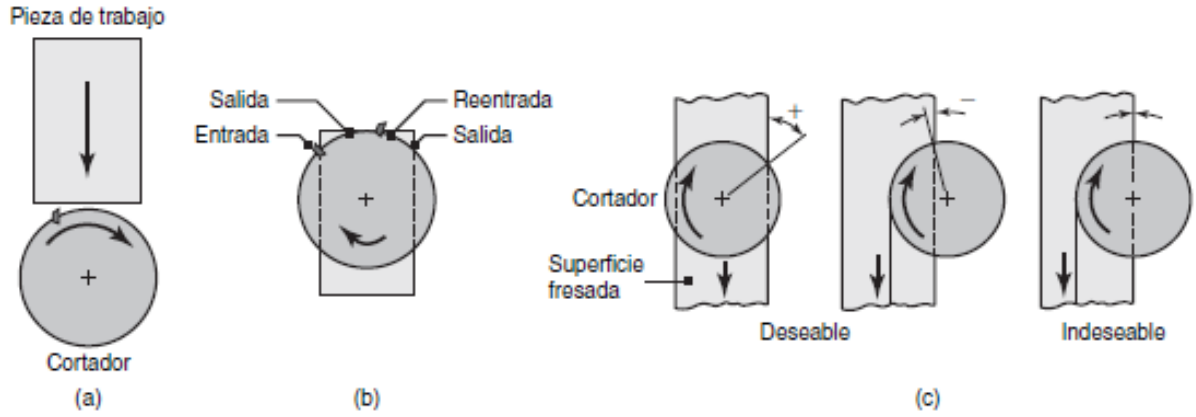
Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.731. ISBN 978-970-26-1026-7

La relación del diámetro del cortador respecto de los ángulos del inserto y su posición relativa respecto de la superficie a fresar es importante porque determina el ángulo al cual un inserto entra y sale de la pieza de trabajo. En la figura 3b de fresado concurrente se observa que (si el inserto tiene ángulos cero de ataque radial y axial, ver figura. 6) la superficie de ataque del inserto ataca directamente la pieza. Sin embargo, en la figura 8a y b se advierte que el mismo inserto puede atacar la pieza a diferentes ángulos, dependiendo de las posiciones relativas del cortador y la anchura de la pieza de trabajo. En la figura 8a se ve que la punta del inserto hace el primer contacto, de manera que existe la posibilidad de que se astille el filo de corte. Por otro lado, en la figura 8b, los primeros contactos (en la entrada, reentrada y las dos salidas) se encuentran en ángulo y lejos de la punta del inserto. Por lo tanto, existe una menor tendencia a que el inserto falle, ya que las fuerzas en el inserto varían con más lentitud. De acuerdo con la figura 6, los ángulos de ataque radial y axial también tienen efecto sobre esta operación.

En la figura 8c se muestran los ángulos de salida de varias posiciones del cortador. Nótese que en los primeros dos ejemplos el inserto sale de la pieza de trabajo en un ángulo, provocando que la fuerza en el inserto se reduzca a cero a una velocidad

menor (deseable) que en el tercer ejemplo, en el que el inserto sale de la pieza de manera repentina (lo cual es poco deseable para la vida de la herramienta).

Figura 8. Posiciones del cortador



(a) Posición relativa del cortador e inserto al entrar en contacto inicialmente con la pieza de trabajo en el fresado de careado. *(b)* Posiciones del inserto hacia el extremo del corte. *(c)* Ejemplos de ángulos de salida del inserto, que muestran posiciones deseables (ángulo positivo o negativo) e indeseables (ángulo cero). En todas las figuras, el husillo del cortador está perpendicular a la página y gira en el sentido de las manecillas del reloj.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.731. ISBN 978-970-26-1026-7

1.3 FRESADO FRONTAL

El fresado frontal es una operación de maquinado importante y común debido a su versatilidad y capacidad para producir diversos perfiles y superficies curvadas. El cortador, conocido como fresa frontal (figura 1c), tiene un zanco recto (para dimensiones pequeñas) o un zanco cónico (para dimensiones más grandes) y se monta en el husillo de la fresadora.

Las fresas frontales se pueden fabricar con aceros de alta velocidad o con insertos de carburo, similares a los del fresado de careado. Por lo general, el cortador gira

en un eje perpendicular a la superficie de la pieza de trabajo y también se puede inclinar para maquinar superficies cónicas o curvadas.

Existen fresas frontales con extremos hemiesféricos (*fresas de punta esférica*) para la producción de superficies esculpidas, como en matrices y moldes. Las *fresas frontales huecas* tienen dientes internos de corte y se utilizan para maquinar la superficie cilíndrica de piezas de trabajo sólidas y redondas. El fresado frontal puede producir varias superficies a cualquier profundidad, como las de tipo curvado, escalonadas y de cavidades (figura 1d). El cortador puede retirar material con su filo de corte del extremo y con las horizontales, como se puede ver en la figura 1c. Las máquinas de husillo vertical y husillo horizontal, así como los centros de maquinado, pueden emplearse para piezas de trabajo de fresado frontal de diversas formas y tamaños. Las máquinas pueden programarse de manera que el cortador siga una serie compleja de trayectorias que optimicen toda la operación de maquinado, a fin de mejorar la productividad y obtener un costo mínimo.

1.3.1 Fresado frontal de alta velocidad. El *fresado frontal de alta velocidad* se ha convertido en un proceso importante con diversas aplicaciones, como el fresado de componentes grandes de aluminio para aeronaves y estructuras de panal con velocidades de husillo de 20,000 a 60,000 rpm. Las máquinas deben tener alta rigidez y precisión, requiriendo por lo general rodamientos hidrostáticos (de aire) y dispositivos de sujeción de trabajo de alta calidad. Los husillos poseen una precisión rotatoria de 10 mm, por lo que la superficie de la pieza de trabajo también es muy precisa. A velocidades tan altas de remoción de material, la recolección y el desecho de las virutas pueden representar un problema significativo.

La producción de cavidades en matrices para conformado de metales (estampado de matrices), como en el forjado o el formado de láminas metálicas, también se efectúa mediante fresado frontal de alta velocidad, utilizando a menudo fresas frontales de punta esférica con recubrimiento de TiAlN (figura 9). Las máquinas tienen movimientos de cuatro o cinco ejes (por ejemplo, ver figura 17) y pueden

manejar matrices hasta de 3 m X 6 m (9 pies X 18 pies) con un peso de 60 toneladas. Así, no es de sorprender que dichas matrices lleguen a costar alrededor de 2 millones de dólares. Las ventajas de las máquinas de cinco ejes son que (a) tienen la capacidad de maquinar formas muy complejas en un solo arreglo; *(b)* pueden emplear herramientas de corte más cortas (reduciendo así la tendencia a la vibración), y *(c)* permiten taladrar orificios a varios ángulos compuestos.

Figura 9. Fresas frontales de punta esférica.

Estos cortadores tienen la capacidad de producir contornos elaborados y con frecuencia se utilizan en el maquinado de matrices y moldes.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.733. ISBN 978-970-26-1026-7.

2. PROGRAMACION LINEAL

En los últimos 70 años, las empresas, cada vez mayores y complejas, han originado una cierta clase de problemas de optimización, donde el interés radica en asuntos tales como la manera más eficiente de manejar una economía, o cómo organizar los horarios de vuelo de las azafatas en una compañía aérea, o la mezcla de ingredientes de un fertilizante para satisfacer las especificaciones agrícolas a un costo mínimo, etc. El estudio de cómo formular y resolver tales problemas ha originado el desarrollo de nuevas e importantes técnicas de optimización. Entre éstas se encuentran la programación lineal[2] . El modelo de programación lineal, esto es, la optimización de una función lineal sujeta a restricciones lineales, es sencillo en su estructura matemática, pero poderoso por su capacidad de adaptarse a un amplio rango de aplicaciones a problemas de la vida real.

Los problemas de programación lineal se interesan en la asignación eficiente de recursos limitados con el ánimo de alcanzar objetivos deseados. Estos problemas se caracterizan por el gran número de soluciones que satisfacen las condiciones impuestas por cada problema. La selección de una solución concreta, como la mejor a un problema, dependerá de cierto objetivo implícito en el planteamiento del problema. Una solución que satisfaga todas las condiciones del problema y además alcance el objetivo deseado se denomina “solución óptima”.

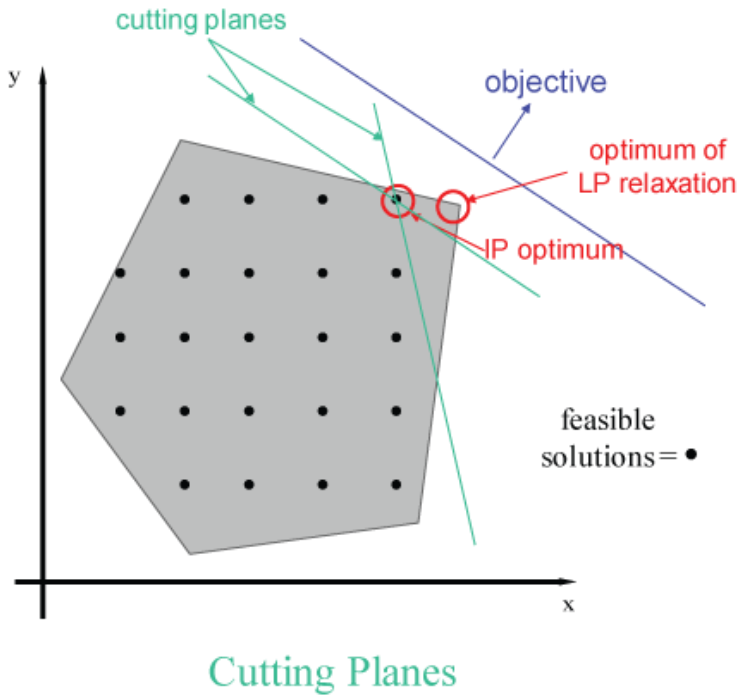
[2] UNIVERSIDAD DE MURCIA. Programación Lineal [En línea]. Disponible en <http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf> (Recuperado en 30 septiembre 2017.)

2.1 ELEMENTOS QUE DEFINEN UN PROBLEMA DE PROGRAMACIÓN LINEAL.

Entre los elementos que definen un problema de programación lineal se puede encontrar:

2.1.1 Definición. Un problema de programación lineal es un problema de minimizar o maximizar una función lineal en presencia de restricciones lineales del tipo desigualdad, igualdad o ambas.

Figura 10. Esquema de Optimización por programación lineal



Fuente: GUROBI OPTIMIZATION. [sitio web]. Mixed-Integer Programming (MIP). (Recuperado en 15 Octubre 2017). Disponible en: http://www.gurobi.com/resources/getting-started/mip-basics

Expresión general de un modelo de programación lineal:

$$\begin{cases} optimizar & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ & sujeto\ a \\ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n (\leq, =, \geq)\ b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n (\leq, =, \geq)\ b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n (\leq, =, \geq)\ b_m \\ x_1, x_2, \cdots, x_k & \geq \quad 0 \quad (k \leq n) \end{cases}$$

Donde optimizar puede ser maximizar o minimizar. *cj* , *bi* y *aij* (1 £ *i* £ m ; 1 £ *j* £ n) son constantes determinadas por la tecnología del problema y *xj* (*j* = 1,2,…,n) son las variables de decisión. Únicamente un signo (£, = o ³) ocurre para cada restricción.

Algunas de las variables de decisión son declaradas no negativas, más adelante se mostrara que cada variable irrestricta puede convertirse equivalentemente en variables no negativas.

La restricción de no negatividad, para todas las variables de decisión, es esencial para el desarrollo del método de solución de los problemas de programación lineal.

Una forma de ver la definición del problema es:

Un problema de Programación Lineal tiene la siguiente forma.

Max *z* =*cx* (función objetivo)

s.a:

Ax = *b* (restricciones funcionales)

x ≥ 0 (restricciones de no negatividad)

2.1.2 Definiciones básicas. Se considera el siguiente problema de programación lineal:

$$\begin{cases} \text{Minimizar } c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \quad \text{sujeto a} \\ \qquad a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ \qquad a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \qquad \vdots \quad \vdots \quad \vdots \quad \vdots \\ \qquad a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \\ \qquad x_1 \ , \ x_2 \ , \cdots , \ x_n \geq 0 \end{cases}$$

La expresión $c_1x_1 + c_2x_2 + \ \ldots + c_nx_n$ recibe el nombre de "*función objetivo*", debe minimizarse y se suele denotar por *z*. Los coeficientes $c_1, c_2, \ldots, c$ de la función objetivo son llamados "*coeficientes de costo*" y $x_1, x_2, \ldots, x_n$ son las "*variables de decisión*" que deben determinarse. Las desigualdades se referencian haciendo uso del orden que ocupan, así $a_{i1} + a_{i2} + \ \ldots + a_m x_n \geq b$ denota la "*i-ésima restricción*". Los coeficientes *ij a* (*i* = 1,2,…,m ; *j* = 1,2,…,n) se llaman los "*coeficientes tecnológicos*". Estos coeficientes forman la siguiente "*matriz de restricciones*", A.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = \left(a_{ij}\right)$$

El vector columna, $b = \begin{Bmatrix} b_1 \\ \vdots \\ b_m \end{Bmatrix}$ al cual se le llama el "*vector del lado derecho*", representa los requerimientos mínimos que deben satisfacerse (es decir, la limitación de los recursos a repartir entre las actividades), por eso a este vector también se le llama "*vector de requerimientos*".

Las restricciones $x_1, x_2, \dots, x_n \geq 0$ son las "*restricciones de no negatividad*".
Cualquier concreción de valores para las variables de decisión $(\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n})$ se llama "solución" (sin importar si es una elección deseable, o incluso admisible).

2.2 SIMPLIFICACIÓN DEL MODELO MATEMÁTICO.

Una vez establecido correctamente el modelo matemático de nuestro problema, se deberá tener en cuenta una serie de consideraciones que simplificarán nuestro trabajo:

2.2.1 Eliminación de restricciones redundantes. Si alguna restricción está incluida en otra, es lógico pensar en su anulación.
Por ejemplo, si se tiene las siguientes restricciones:

(X1/a11) + (X2/b11) <=1

(X1/a21) + (X2/b21) <=1

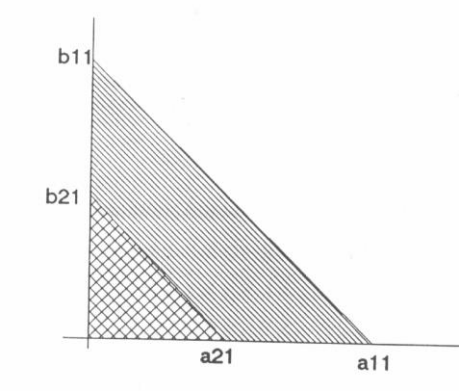
Si se cumple:

A11/a21>1

B11/b21>1

Gráficamente se representa en la figura.

Figura 11. Representación de restricciones



Fuente: CABRERA RODRÍGUEZ, Sonia I. Aplicación de la programación Lineal a la agronomía. [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en: http://matematicas.uclm.es/ita-cr/web_matematicas/trabajos/248/Programacion_lineal.pdf

Nuestra función objetivo (F.O.) trata un problema de maximización; es evidente que se podrá eliminar la restricción.

$$(X1/a11) + (X2/b11) <= 1$$

2.2.2 Eliminación de restricciones obvias.

Dado que la restricción de para todo i: Xi >=0 va implícita en nuestros problemas, se pueden obviar las restricciones del tipo:

$$a*X1 + b*X2 >=0$$

2.2.3 Eliminación de variables inútiles. Si al modelizar nuestro problema, en la función objetivo, aparecen variables no sujetas a restricciones (variables que no aparecen en las restricciones),se podrá eliminar de nuestro problema, ya que el valor de estas variables no condiciona la solución del problema.

2.2.4 División en subproblemas. Si al analizar un problema, se observa que se pueden dividir las restricciones en conjuntos distintos (de tal forma que no tengan variables comunes) también se podrá dividir el problema en tantos subproblemas como conjuntos de restricciones se tengan. La solución del problema original será la unión de las soluciones de los subproblemas tratados.

Se pondrá un ejemplo.
La modelización de un problema es:

F.O...: Max 2 X1 + 4 X2 + 3 X3 + 8 X4

S.a...: X1 + 9 X2 <=7

5 X1 + 7 X2 >=9

2 X3 + X4 <=3

4 X3 + 6 X4 >=12

Se podrá dividir nuestro problema en subproblemas:

A) F.O...: Max 2 X1 + 4 X2

S.a...: X1 + 9 X2 ≤7

5 X1 + 7 X2 ≥9

B) F.O...: Max 3 X3 + 8 X4

S.a...: 2 X3 + X4 <=3

4 X3 + 6 X4 >=12

Nota: Es lógico pensar que si en el problema se tienen n variables y n restricciones linealmente independientes y sin contracciones (que no estén unas contenidas en otras), sólo existirá una única solución al problema.

2.2.5 Homogeneización de restricciones. Como se verá más adelante, para poder resolver los problemas de programación lineal por el método Simplex, será conveniente tener las restricciones del problema de tal forma que los términos “bi” sean mayores o iguales a cero.
Por ello, ya que se puede encontrar con restricciones del tipo:

$$\Sigma\, ai * Xi \geq -bi$$

$$\Sigma\, ai * Xi \leq -bi$$

$$\Sigma\, ai * Xi = -bi$$

Se podrá homogeneizar nuestro sistema, convirtiéndolo al tipo:

$$\Sigma\ -ai * Xi \leq bi$$

$$\Sigma\ -ai * Xi \geq bi$$

$$\Sigma\ -a * Xi = bi$$

Con sólo multiplicar por –1 y cambiar el sentido de la desigualdad. Es decir:

$$\Sigma\, ai * Xi \geq -bi \rightarrow \Sigma\,(-1) * ai * Xi \leq bi$$

Donde se mantendrán las condiciones de la restricción y se posibilitará la resolución del problema mediante el método Simplex.

2.3 TIPOS DE RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN LINEAL.

2.3.1 Resolución gráfica de problemas de programación lineal. Los problemas de programación lineal con dos variables de decisión se pueden resolver fácilmente de forma gráfica. El método consiste en representar gráficamente la región factible del problema para después introducir en dicho gráfico las curvas de nivel de la función objetivo del problema.

Se observara esto resolviendo gráficamente el problema del ejemplo de la cerveza.

El problema que se tiene que resolver es el siguiente:

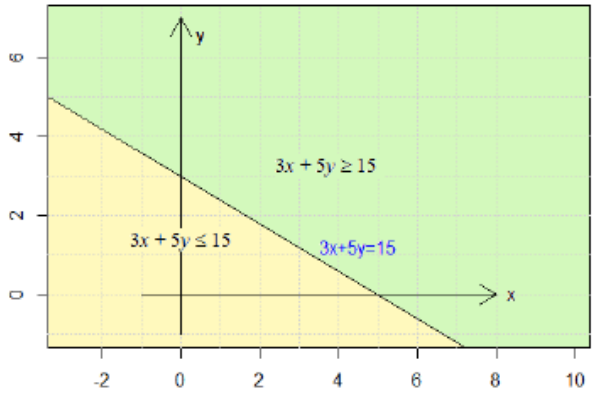
$$(P)\begin{cases} \text{Maximizar } z = 5x + 3y \\ \text{sujeto a} \\ 3x + 5y \leq 15 \\ 5x + 2y \leq 10 \\ x\ ,\ y\ \geq 0 \end{cases}$$

La región factible de este problema, *F*, es el conjunto de todos los puntos (*x*,*y*) de R2 que cumplan, de forma simultánea, todas las restricciones del problema. Es decir, el siguiente conjunto de R2:

$$F\ \{(x,y) \in R2/3x + 5y\ \leq 15, 5x + 2y\ \leq 10, x\ \geq 0, \qquad y\ \geq 0\}$$

Para representar gráficamente este conjunto de R2 se hará lo siguiente: Se identifica el espacio R2 con un plano, la hoja de papel. En este plano se dibujaron unos ejes cartesianos (*x*, *y*). Para ver los puntos del plano que cumplen la primera restricción, 3*x*+5*y* =15, se observa que la ecuación lineal, 3*x*+5*y* <=15, es la ecuación de una recta en el plano. Dicha recta divide al plano en dos semiplanos, uno a cada lado de la recta. Como se puede observar en la figura 20:

Figura 12. Representación R^2

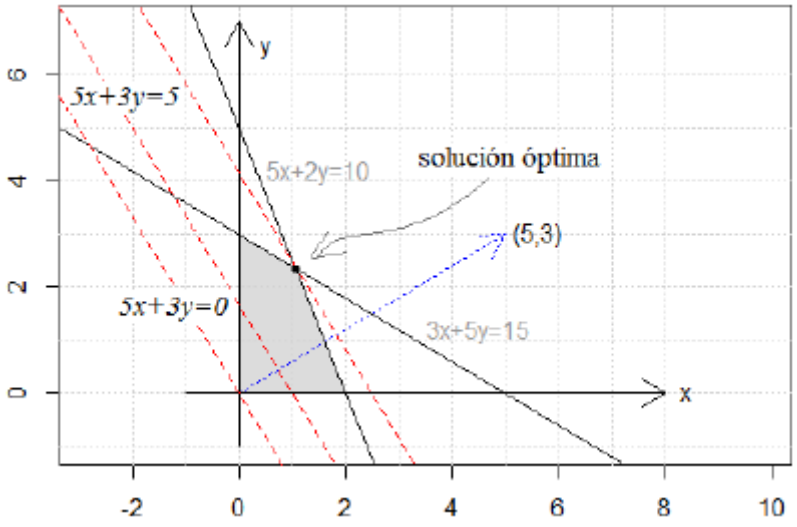


Fuente: UNIVERSIDAD DE MURCIA. Programación Lineal [En línea]. Disponible en:http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf (Recuperado en 30 septiembre 2017.)

Una vez que se ha representado la región factible, el problema consiste en encontrar el punto, o puntos, de la misma que hace máximo el valor de la función objetivo, *z* =5*x* +3*y*. Para averiguar esto sería interesante que se pudiera introducir a la función objetivo dentro de la representación gráfica que se está haciendo del problema. Esto

se puede conseguir del siguiente modo. Se observara que si se coge la expresión de la función objetivo, 5*x* +3*y,* y se iguala a una constante, *k*, se obtendría la ecuación de una recta en el plano, 5*x* +3*y* =*k,* ocurriendo que todos los puntos del plano que están sobre esa recta proporcionan el mismo valor en la función objetivo del problema, es decir, en todos ellos ocurrirá que z será igual a *k*. Por otro lado, todas las rectas de ecuación 5*x* +3*y* =*k*, cuando *k* es un número cualquiera, son paralelas entre sí, con vector perpendicular a todas ellas, el vector (5,3). Además dicho vector, (5,3), proporciona la dirección en la cual, si se trasladan la rectas paralelas en dicha dirección, ocurre que el valor de *k* aumenta. Esto indica una forma de proceder. Se puede empezar por dibujar la recta, 5*x* +3*y* =0, e ir dibujando rectas paralelas a ella, en la dirección indicada por el vector (5,3), al hacer esto se ve que llegará un momento en que tales rectas dejarán de cortar a la región factible y además por pura observación se podrá observar en qué punto (o puntos) se despegarían estas rectas de la región factible, tal como se puede apreciar en la Figura 13:

Figura 13. Rectas de región factible



Fuente: UNIVERSIDAD DE MURCIA. Programación Lineal [En línea]. Disponible en:http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf (Recuperado en 30 septiembre 2017.)

Luego, a la vista del dibujo, se ve que la solución óptima del problema será el punto de corte de las rectas, 3*x* +5*y* =15 y 5*x* +2*y* =10. Es decir, el máximo de la función objetivo sobre la región factible *F* se obtiene en el punto Q (20/19,45/19), que es el punto de intersección de tales rectas. El valor que toma la función objetivo en la solución óptima del problema es:

$$5\frac{20}{19} + 3\frac{45}{19} = \frac{235}{19} = 12.36842$$

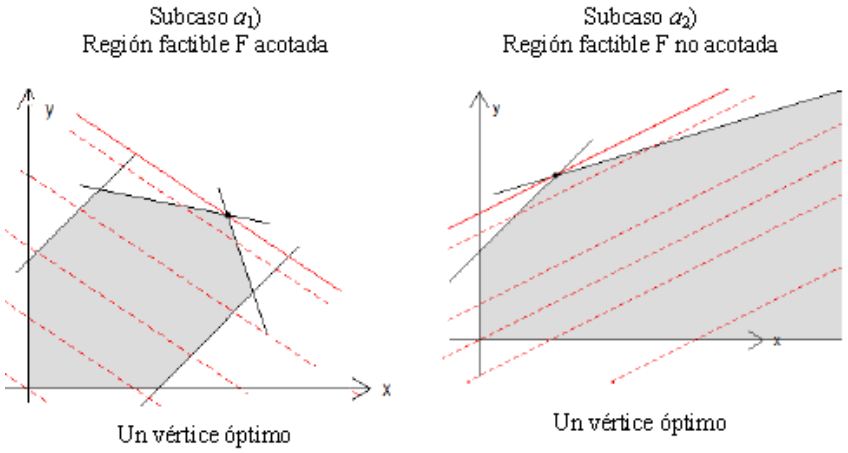
Los Casos posibles:

Se verá con diversos ejemplos todas las situaciones que pueden presentarse respecto a la región factible de un problema de programación lineal; así como respecto al alcance del valor óptimo de la función objetivo en uno, ninguno o infinitos puntos.

Se supone que los gráficos que vienen a continuación representan la región factible, F, en el espacio de las variables de decisión, suponiendo que hay dos variables de decisión, y que los problemas son de maximizar en forma canónica.

Caso *a*): Óptimo único finito. Se pueden dar dos subcasos

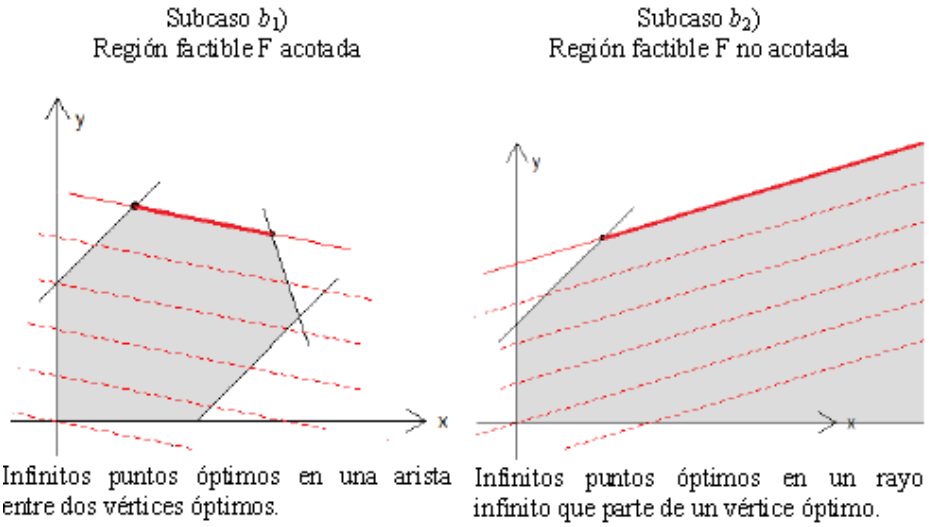
Figura 14. Subcaso único finito



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

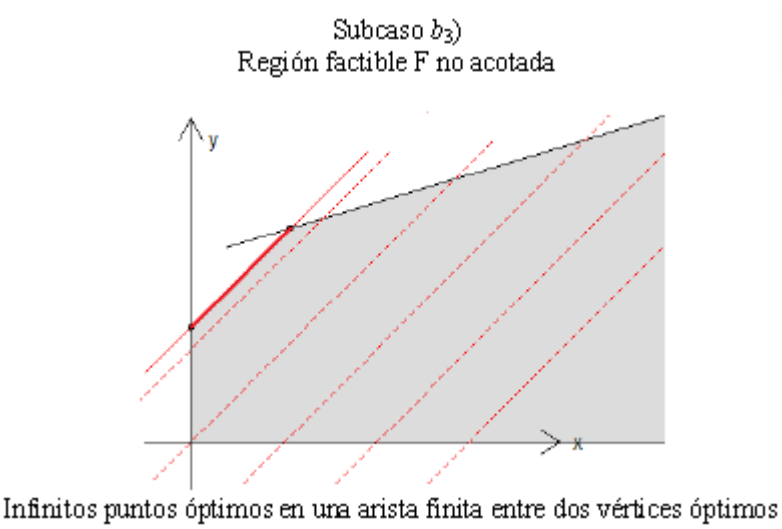
Caso b): Infinitos puntos de óptimo finito: Se pueden dar tres subcasos

Figura 15. Casos de puntos de óptimos finitos



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

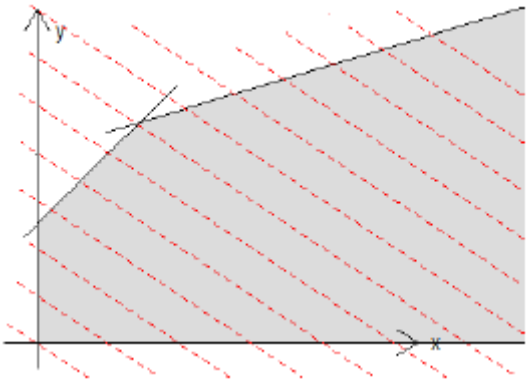
Figura 16. Subcaso b_3



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

Caso *c*): Valor objetivo óptimo no acotado. Este caso sólo puede darse si la región factible, F, es no acotada.

Figura 17. Región no acotada



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

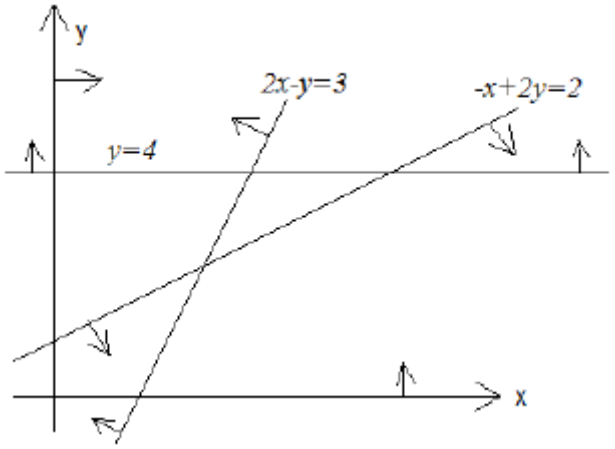
Como se puede apreciar en el dibujo el valor de la función objetivo no está acotado en la región factible. Es decir, dado cualquier número real positivo, por grande que sea, se pueden encontrar soluciones factibles del problema donde el valor de la función objetivo supera al número dado.

Caso *d*): Región factible, F, vacía.

Sea el siguiente problema de programación lineal:

$$(P)\begin{cases} \text{Maximizar } z = 2x - 3y \\ \text{sujeto a} \\ 2x - y \le 3 \\ -x + 2y \le 2 \\ y \ge 4 \\ x\ ,\ y\ \ge 0 \end{cases}$$

Figura 18. Representación de la región factible



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

Como se puede observar, la región factible es vacía. Es decir, el problema no tiene soluciones factibles.

Observando los casos que se acaban de ver se aprecia que cuando un problema de programación lineal tiene solución óptima finita, aunque haya infinitas de ellas, siempre hay al menos un vértice óptimo. Esto es un hecho general, que se puede demostrar.

Caso e) El problema del viajante.

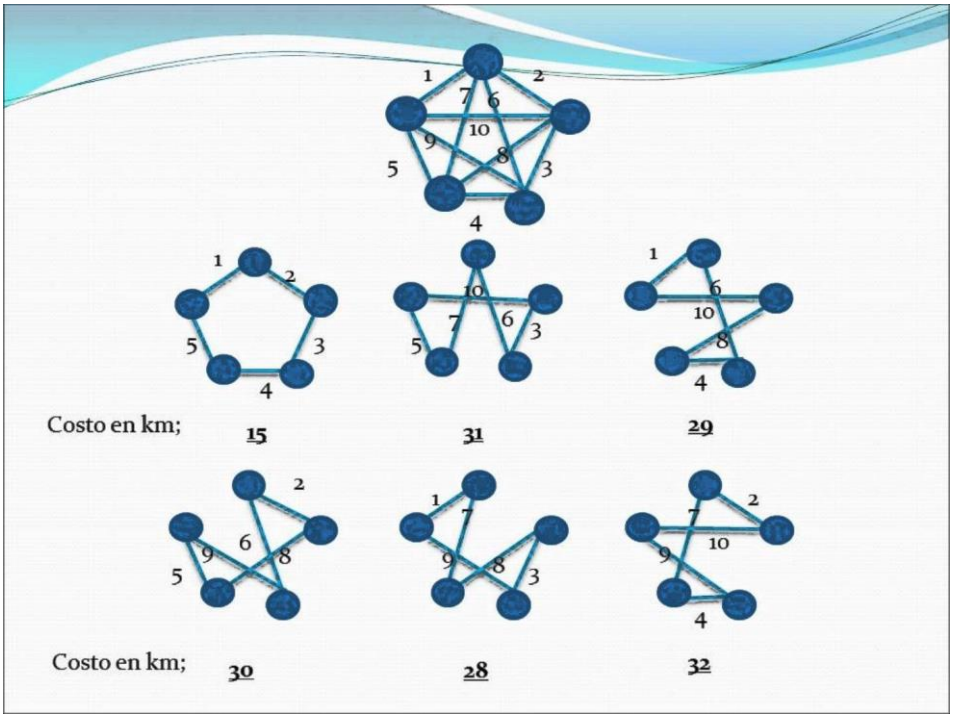
Figura 19. Agente viajero

Fuente: [sito web]. Estudio de heurísticas para el problema del Agente Viajero Asimétrico. (Recuperado en 15 Octubre 2017). Disponible en:http:/www.dane.gov.co/index.php/construcción-alias/indicadores-economicos-alrededor-de-la-construcion-ieac

Un viajante debe visitar las ciudades 1*; 2;…; n* partiendo de la ciudad 0, pasando por cada una de las ciudades 1*; 2;…; n* una y solo una vez y volviendo luego a la ciudad 0. Sea *cij* el costo de viajar de la ciudad *i* a la ciudad j $(0 \leq i, j \leq n)$. Si no hay camino de *i* a *j se* toma *cij* = ∞.

A cada una de las posibles maneras de hacer el recorrido se llama un circuito Hamiltoniano figura 20. Se define el costo de un tal circuito como la suma de los costos de los tramos que lo componen. El problema consiste en hallar un circuito Hamiltoniano de mínimo costo.

Figura 20. Esquema de Circuito Hamiltoniano



Fuente: [sito web]. (Recuperado en 15 Octubre 2017). Disponible en: https://www.youtube.com/watch?v=stQhPKdipoc

Se representa la situación en un grafo dirigido completo *G* = (*V*,*E*) donde

$$V = \{0,1,2,\ldots,n\}$$

Y se asigna costo *cij* a cada rama (*i; j*).

Dado un circuito Hamiltoniano *C* (es decir, un circuito dirigido en *G* que pasa por cada vértice una y solo Una vez), para cada (*i; j*) sea

$$\delta = \begin{cases} 1 & si (i,j) \epsilon\ C \\ 0 & si\ no \end{cases}$$

Entonces el costo de *C* es $\sum Cij\ \delta ij$

Se observa que para cada vértice *i* hay una sola rama de *C* cuya cola es *i* y una sola rama de *C* cuya punta es *i*. Luego se satisface

$$\sum_j \delta_{ij} = 1 \ \ (0 \leq i \leq n)$$

(8)

$$\sum_i \delta_{ij} = 1 \ \ (0 \leq j \leq n)$$

Se supondrá ahora que para cada (*i, j*) se tiene definido un $\delta ij \ tal \ que \ \delta ij = 0 \ o \ \delta ij = 1$ y de manera tal que valga (8).

Si el conjunto de ramas (*i; j*) tales que *δij* = 1 fuese un circuito entonces podríamos pensar a cada circuito Hamiltoniano como una solución factible de

$$\sum_j \delta_{ij} = 1 \ \ (0 \leq i \leq n)$$

$$\sum_i \delta_{ij} = 1 \ \ (0 \leq j \leq n) \quad (9)$$

$$0 \leq \delta ij \leq n \quad \delta ij \ entero$$

Cuyo costo es $\sum Cij \ \delta ij$ Lamentablemente esto no es así, como lo muestra el siguiente ejemplo:

Sea *V* = *{*0*;* 1*;* 2*;* 3*;* 4*;* 5*}* y se toma

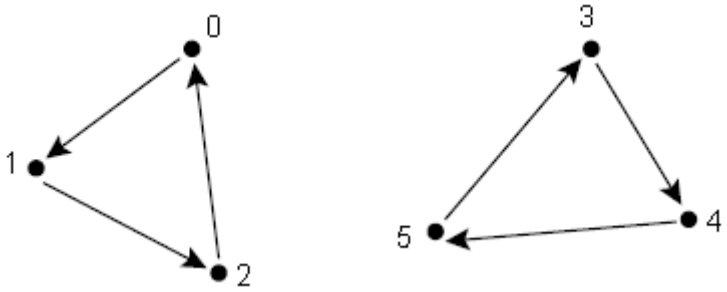
δ01 = δ20 = δ12 = δ34 = δ45 = δ53 = 1

Y los restantes δ*ij* iguales a cero. Es decir, el conjunto (*i; j*) tales que δ *ij* = 1 es

{(0*;* 1)*;* (2*;* 0)*;* (1*;* 2)*;* (3*;* 4)*;* (4*;* 5)*;* (5*;* 3)*}*

En este caso δ*ij* (0 ≤ *i, j* ≤ 5) es una solución factible de (9) pero el conjunto de ramas (*i, j*) tales que δ*ij* = 1 no forman un circuito sino dos subcircuitos:

Figura 21. Subcircuitos



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.dm.uba.ar/materias/investigacion_operativa/2011/2/capit_5.pdf

Pero agregando la ingeniosa condición

$$ui\ uj\ +\ n\delta ij\ \leq\ n\quad 1\,(1\ \leq\ i,j\ \leq\ n,i\ /\ j)$$

Donde $ui\ (1\ \leq\ i\ \leq\ n)$ son números reales nos permitirá plantear el problema del viajante por programación lineal entera. Esto se debe a que si δ*ij* es una solución factible de (9) entonces existen *ui* satisfaciendo esta nueva condición si y solo si el conjunto de ramas (*i; j*) tales que δ*ij* = 1 es un circuito. En tal caso el costo de ese circuito es $\sum Cij\ \delta ij$

Afirmación. El problema de programación lineal entera

$$\begin{aligned} &min \textstyle\sum_i c_{ij}\delta_{ij} \\ &\textstyle\sum_j \delta_{ij} = 1\ \ (0 \leq i \leq n) \qquad (10)\\ &\textstyle\sum_i \delta_{ij} = 1\ \ (0 \leq j \leq n) \\ &ui - uj + n\delta ij \leq n-1 \quad (1 \leq i,j \leq n, i \neq j) \\ &0 \leq \delta ij \leq 1 \quad \delta ij\ entero \end{aligned}$$

Resuelve el problema del viajante.

Demostración: Se mostrara que cada circuito Hamiltoniano puede representarse como una solución factible de (10).

Se supone que *C* es un circuito Hamiltoniano. Para cada (*i; j*) sea

$$\delta = \begin{cases} 1 & si (i,j) \epsilon\ C \\ 0 & si\ no \end{cases}$$

Sea *ui* = 1 si *i* es la primera ciudad visitada, *ui* = 2 si *i* es la segunda ciudad visitada, etc. Por ejemplo, si *n* = 5 y el circuito es

$$0 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 0$$

Entonces

$$u1 = 4;\ u2 = 1;\ u3 = 3;\ u4 = 5;\ u5 = 2$$

Entonces

$$ui - uj + n\delta ij = \begin{cases} ui - uj & si (i,j) \notin\ C \\ -1 + n & si (i,j) \in\ C \end{cases}$$

Ya que si (*i, j*) $\notin$ *C* entonces *δij* = 0 y si (*i, j*) $\notin$ *C* entonces el viajante visita *j* justo después de visitar *i*, de donde *uj* = *ui* + 1 y como (*i, j*) ϵ *C* entonces δ*ij* = 1.

Luego, como 1 ≤ *ui* ≤ *n* entonces *ui* $\neq$ *uj* + *nδij* ≤ *n*-1 para todo $i \neq j\ (1 \leq i;\ j \leq n)$.

Esto muestra que cada circuito Hamiltoniano determina una solución factible de (10).

Recíprocamente, se supone ahora que se tiene una solución factible de (10). Se probara que el conjunto de ramas *C* = {(*i; j*) / *δij* = 1*}* es un circuito Hamiltoniano. Como cada *δij* es cero o uno y valen

$$\sum_j \delta_{ij} = 1\ \ (0 \leq i \leq n)$$

$$\sum_i \delta_{ij} = 1\ \ (0 \leq j \leq n)$$

Entonces para cada *i* hay un único *j* tal que *δij* = 1 y un único *j* tal que δ*ji* = 1 (notese que la segunda igualdad puede escribirse $\sum_i \delta_{ij} = 1\ \ (0 \leq i \leq n)$ Esto significa que

para cada vértice *i* hay una sola rama de *C* cuya cola es *i* y una sola rama de *C* cuya punta es *i*. Luego, solo se debe ver que las ramas de *C* forman un circuito y no dos o más subcircuitos. Se supone que formaran dos o más subcircuitos. Entonces se puede elegir uno de ellos que no pase por el vértice 0. Sea *E* el conjunto de ramas que lo forman y sea *k* # ε. Como *ui* *uj* + *nδij* ≤ *n* - 1 para todo 1 ≤ *i, j* ≤ *n* tal que *i* / *j* entonces

$$\sum_{(i,j)\in\varepsilon} u_i - u_j + n\delta_{ij} \leq k(n-1)$$

Pero como para cada *i* hay una única rama en *E* cuya cola es *i* y una única rama cuya punta es *i* entonces

$$\sum_{(i,j)\in\varepsilon} u_i - u_j = 0$$

Ya que cada *ui* aparece una vez sumando y una vez restando. Además, como las ramas de *E* forman un subcircuito de *C* entonces δ*ij* = 1 para todo (*i; j*) ε *E*. Por lo tanto,

$$\sum_{(i,j)\in\varepsilon} n\delta_{ij} = kn$$

De donde

$$kn = 0 + kn = \sum_{(i,j)\in\varepsilon} u_i - u_j + \sum_{(i,j)\in\varepsilon} n\delta_{ij} = \sum_{(i,j)\in\varepsilon} u_i - u_j + n\delta_{ij} \leq k(n-1)$$

Con lo cual se tiene que *kn* ≤ *k*(*n* - 1), lo que es un absurdo pues *k* > 0. Luego, cada solución factible de (10) determina un circuito Hamiltoniano.

2.5.2 Programación lineal mixta. Son aquellos en que al mismo tiempo hay variables continuas y variables que solo utilizan valores entero.

2.5.3 Programación lineal binaria. Son aquellas en que la variable solo puede tomar valores de 1 o 0, se usa generalmente en problemas de inclusión o exclusión.

3. REDES NEURONALES

3.1 ANTECEDENTES

El desarrollo de las redes neuronales[3] ha tenido un gran desarrollo a lo largo de la historia, por tanto es preponderante entender un poco como ha sido su evolución y como ha venido cambiando la concepción hasta la sociedad contemporánea, debido a esto se cita el breve panorama histórico de Hilera. Conseguir diseñar y construir máquinas capaces de realizar procesos con cierta "inteligencia" ha sido uno de los principales objetivos y preocupaciones de los científicos a lo largo de la historia. De los intentos realizados en este sentido se ha llegado a definir las líneas fundamentales para la obtención de máquinas inteligentes: En un principio los esfuerzos estuvieron dirigidos a la obtención de autómatas, en el sentido de máquinas que realizaran, con más o menos éxito, alguna función típica de los seres humanos. Pero esto no era más que el resultado del desarrollo técnico de la habilidad mecánica de los constructores de tales artefactos. Sin embargo, en esta misma línea se sigue investigando hoy en día con herramientas enormemente sofisticadas y con resultados realmente sorprendentes: la habilidad mecánica ha pasado a convertirse en disponibilidad microinformática (lenguajes Lisp y Prolog, sistemas expertos, etc.), de forma que, actualmente, existen diversas maneras de realizar procesos similares a los inteligentes y que se puede encuadrar dentro de la denominada Inteligencia Artificial (IA).

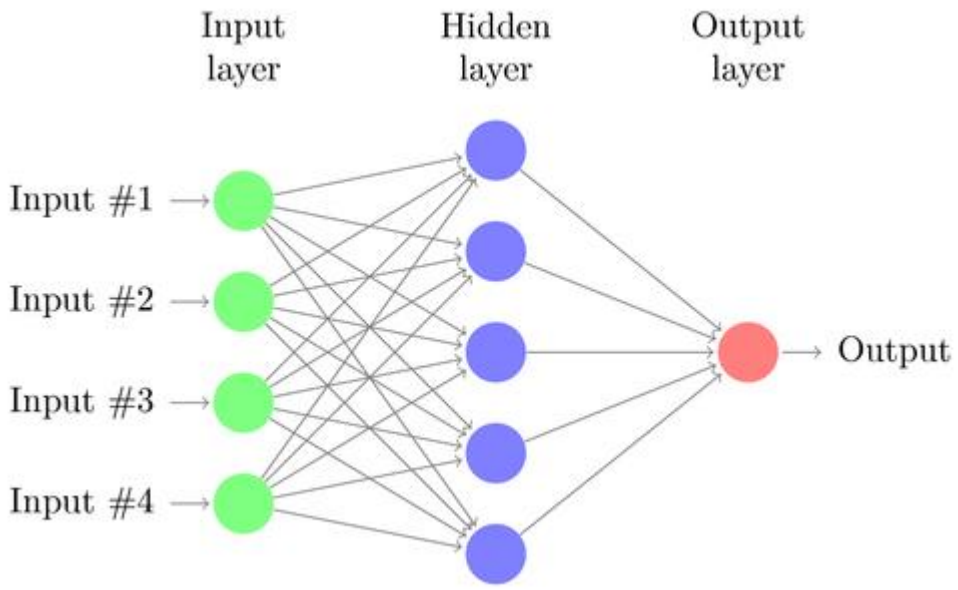
[3] HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. REDES NEURONALES ARTIFICIALES. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995.

3.2 DEFINICIÓN DE RED NEURONAL

Existen numerosas formas de definir lo que son las redes neuronales; desde las definiciones cortas y genéricas hasta las que intentan explicar más detalladamente lo que significa "red neuronal" o "computación neuronal". Se verán algunos ejemplos de ambos casos:

Una nueva forma de computación, inspirada en modelos biológicos.

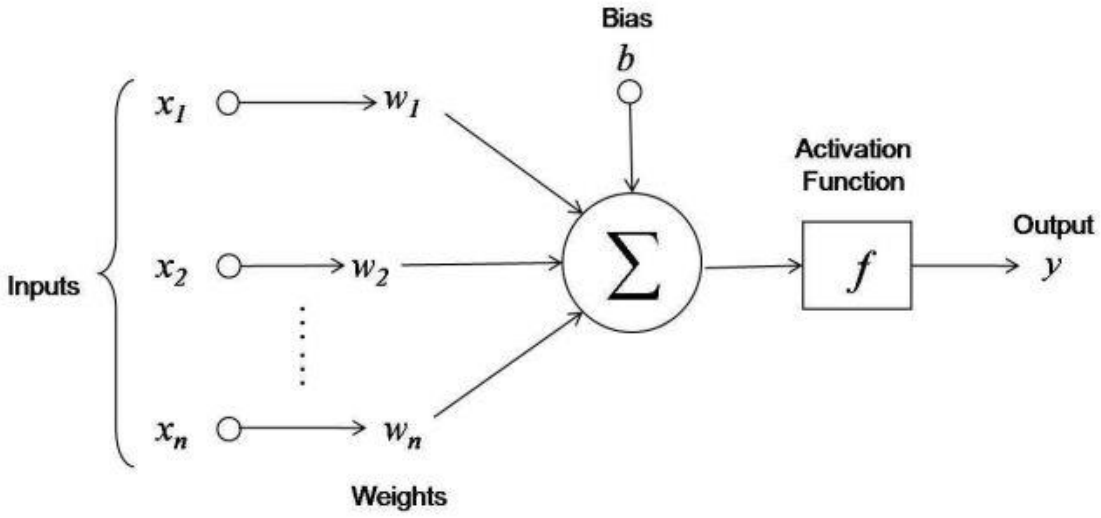
Figura 22. Esquema simple de red neuronal.



Fuente: [sitio web]. (Recuperado en 15 octubre 2017). Disponible en: http://www.texample.net/tikz/examples/neural-network/

Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.

Figura 23. Diagrama de una red neuronal artificial.



Fuente: [sitio web]. (Recuperado en 15 octubre 2017). Disponible en: https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network

... un sistema de computación hecho por un gran número de elementos simples, elementos de proceso altamente interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.

Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

3.3 VENTAJAS DE LAS REDES NEURONALES.

Debido a su constitución y a sus fundamentos, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos

anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante,
etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas. Estas ventajas incluyen:

- Aprendizaje adaptativo. Capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial.

- Autoorganización. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.

- Tolerancia a fallos. La destrucción parcial de una red conduce una degradación de su estructura, sin embargo, algunas capacidades de la red se pueden retener incluso sufriendo un gran daño.

- Operación en tiempo real. Los computadores neuronales pueden ser realizados en paralelo y máquinas con especial hardware se diseñan y fabrican para obtener esta capacidad.

- Fácil inserción dentro de la tecnología existente. Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes.

3.4 EL MODELO HOPFIELD

Sin duda, uno de los principales responsables del desarrollo que ha experimentado el campo de la computación neuronal ha sido J. Hopfield, quien construyó un modelo de red Hopfield con el número suficiente de simplificaciones como para poder extraer analíticamente información sobre las características relevantes del sistema,

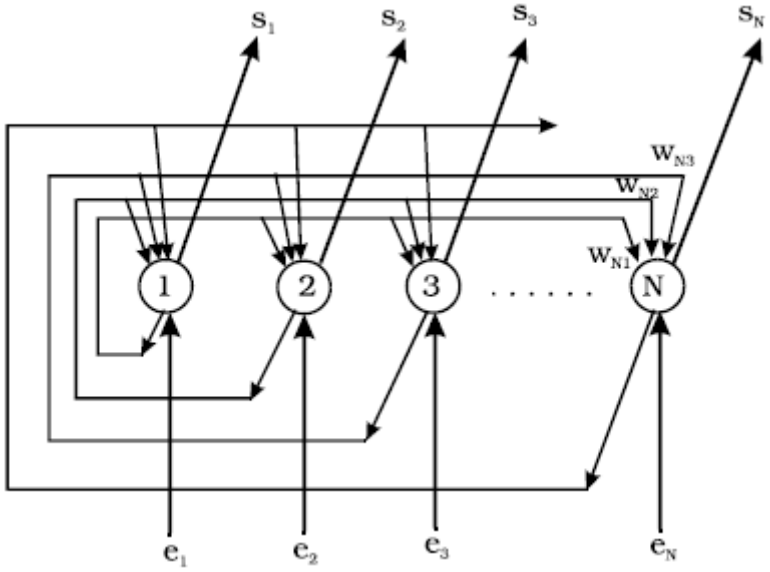
conservando las ideas fundamentales de las redes construidas en el pasado y presentando una serie de funciones básicas de sistemas neuronales reales. Además, Hopfield supo establecer un paralelismo entre su modelo y ciertos sistemas extensamente estudiados en física estadística, lo cual ha permitido aplicar todo un conjunto de técnicas bien conocidas en este campo y, con ello, producir un avance en la comprensión del funcionamiento de las redes neuronales.

Con su aportación, Hopfield redescubrió el mundo casi olvidado de las redes autoasociativas, caracterizadas por una nueva arquitectura y un nuevo funcionamiento, a las que se tuvo que añadir otro tipo de reglas de aprendizaje. Las consecuencias fueron redes con un comportamiento diferente a las diseñadas con estructura feed-foward (Adaline/Madaline, Perceptron,…), analizadas en el capítulo anterior.

3.4.1 Arquitectura El modelo de Hopfield (Figura 24) consiste en una red monocapa con N neuronas cuyos valores de salida son binarios: 0/1 ó -1/+1. En la versión original del modelo (DH: Discrete Hopfield) las funciones de activación de las neuronas eran del tipo escalón. Se trataba, por tanto, de una red discreta, con entradas y salidas binarias; sin embargo, posteriormente Hopfield desarrolló una versión continua con entradas y salidas analógicas, utilizando neuronas con funciones de activación tipo sigmoidal (CH: Continuos Hopfield).

Cada neurona de la red se encuentra conectada a todas las demás (conexiones laterales), pero no consigo misma (no existen conexiones auto-recurrentes). Además, los pesos asociados a las conexiones entre pares de neuronas son simétricos. Esto significa que el peso de la conexión de una neurona "i" con otra neurona "i" es de igual valor que el de la conexión de la neurona "j" con la "j" (W_{ij}= W_{ji}).

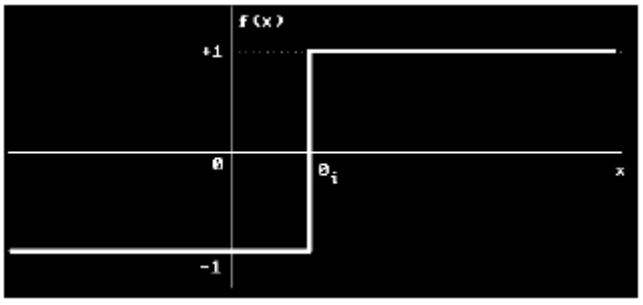
Figura 24.Red Hopfield



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.182.

La versión discreta de esta red fue ideada para trabajar con valores binarios -1 y +1 (aunque mediante un ajuste en los pesos pueden utilizarse en su lugar los valores 1 y 0). Por tanto, la función de activación de cada neurona (i) de la red (f(x)) es de tipo escalón:

Figura 25. Función tipo escalón



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.183.

$$f(x) = \begin{cases} +1, & x > \theta_i \\ -1, & x < \theta_i \end{cases}$$

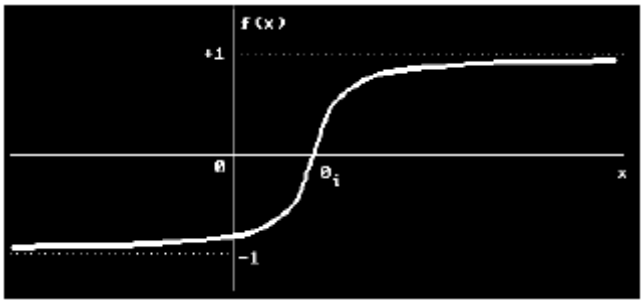
Cuando el valor de x coincide exactamente con θi la salida de la neurona i permanece con su valor anterior. θ_i Es el umbral de disparo de la neurona i, que representa el desplazamiento de la función de transferencia a lo largo del eje de ordenadas (x). En el modelo de Hopfield discreto suele adoptarse un valor proporcional a la suma de los pesos de las conexiones de cada neurona con el resto:

$$\theta_i = k \sum_{j=1}^{N} W_{ij}$$

Si se trabaja con los valores binarios -1 y +1, suele considerarse el valor nulo para θ_i. Si los valores binarios son 0 y 1, se toma un valor de 1/2 para k.

En el caso de las redes de Hopfield continuas, se trabaja con valores reales en los rangos [-1,+1] o [0,1]. En ambos casos la función de activación de las neuronas es de tipo sigmoidal. Si se trabaja con valores entre -1 y +1 la función que se utiliza es la tangente hiperbólica:

Figura 26. Función tangente hiperbólica

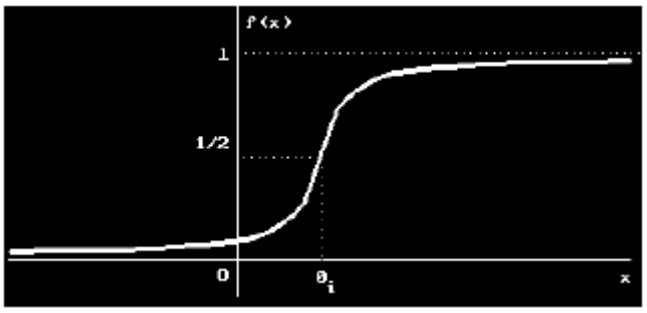


Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.184.

$$f(x-\theta_i) = \text{tgh}(\alpha(x-\theta_i)) = \frac{e^{\alpha(x-\theta_i)} - e^{-\alpha(x-\theta_i)}}{e^{\alpha(x-\theta_i)} + e^{-\alpha(x-\theta_i)}}$$

Si el rango es [0,1], se utiliza la misma función que para la red backpropagation:

Figura 27. Igual función de la red backpropagation



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.184.

$$f(x-\theta_i) = \frac{1}{1 + e^{-\alpha(x-\theta i)}}$$

En ambos casos α es un parámetro que determina la pendiente de la función sigmoidal.

4.3 APRENDIZAJE

La red Hopfield tiene un mecanismo de aprendizaje *Off Line*. Por tanto, existe una etapa de aprendizaje y otra de funcionamiento de la red. En la etapa de aprendizaje se fijan los valores de los pesos en función de las informaciones que se pretende memorice o almacene la red. Una vez establecidos, la red entra en funcionamiento tal y como se describió en el apartado anterior.

Esta red utiliza un aprendizaje no supervisado de tipo Hebbiano, de tal forma que el peso de una conexión entre una neurona "i" y otra "j" se obtiene mediante el producto de los componentes i-ésimo y j-ésimo del vector que representa la información o patrón que debe almacenar. Si el número de patrones a "aprender" es M, el valor definitivo de cada uno de los pesos se obtiene mediante la suma de los M productos obtenidos por el procedimiento anterior, un producto por información a almacenar.

En el caso de la Red Hopfield Discreta que trabaja con valores -1/+1, este algoritmo de aprendizaje puede expresarse de la siguiente forma:

$$W_{ij} = \begin{cases} \sum_{k=1}^{M} e_i^{(k)} e_j^{(k)} & 1 \le i,j \le N \quad ; i \ne j \\ 0 & 1 \le i,j \le N \quad ; i = j \end{cases}$$

Siendo:

W_{ij}: Peso asociado a la conexión entre la neurona i y la neurona j, que coincide con W_{ji}

$e_j^{(k)}$: Valor de la componente i-ésima del vector correspondiente a la información k-ésima que debe "aprender" la red.

N: Número de neuronas de la red y, por tanto, tamaño de los vectores de aprendizaje.
M: Número de informaciones que debe "aprender" la red.
Si la Red trabajase con valores discretos 0/1 en lugar de -1/+1, entonces los pesos se calculan según la expresión:

$$W_{ij} = \begin{cases} \sum_{k=1}^{M} (2e_i^{(k)} - 1)(2e_j^{(k)} - 1) & 1 \le i,j \le N \quad ; i \ne j \\ 0 & 1 \le i,j \le N \quad ; i = j \end{cases}$$

El algoritmo de aprendizaje también se suele expresar utilizando una notación matricial. En tal caso se podría considerar una matriz W de dimensiones *NxN* que representase todos los pesos de la red:

$$W=\begin{bmatrix} w_{11} & w_{21} & w_{31} & \cdots & w_{N1} \\ w_{12} & w_{22} & w_{32} & \cdots & w_{N2} \\ w_{13} & w_{23} & w_{33} & \cdots & w_{N3} \\ & & \vdots & & \\ w_{1N} & w_{2N} & w_{3N} & \cdots & w_{NN} \end{bmatrix}$$

Esta matriz es simétrica, al cumplirse que w_{ij}= w_{ji} y tiene una diagonal principal con valores nulos debido a la no existencia de conexiones auto-recurrentes (w_{ii}=0).

También se tendría el conjunto de los M vectores que representan las informaciones que ha de *aprender* la red:

$$E_1 = [e_1^{(1)}, e_2^{(2)}, \dots, e_N^{(1)}]$$

$$E_2 = [e_1^{(2)}, e_2^{(2)}, \dots, e_N^{(2)}]$$

$$\dots$$

$$E_M = [e_1^{(M)}, e_2^{(M)}, \dots, e_N^{(M)}]$$

Utilizando esta notación, el aprendizaje consistiría en la creación de la matriz de pesos *W* a partir de los *M* vectores o informaciones de entrada (E_1,…,E_M) que se *enseñan* a la red. Matemáticamente se expresaría:

$$W = \sum_{k=1}^{M} [E_k^T E_k - I]$$

Donde la matriz E_k^T es la traspuesta de la matriz E_k, e I es la matriz identidad de dimensiones *NxN* que anula los pesos de las conexiones auto-recurrentes (w_{ii}).

3.4.2.1 La función energía. Como ya se ha comentado, el aprendizaje de la red Hopfield es de tipo Hebbiano. La elección de esta regla de aprendizaje por Hopfield fue, entre otras razones, debido a que asegura la estabilidad de la red, es decir, la convergencia hacia una respuesta estable cuando se presenta una información de entrada.

Muchas de las investigaciones acerca de la estabilidad de las redes se basan en el establecimiento de una función, denominada Función Energía de la Red, para representar los posibles estados (puntos de equilibrio) de la red. De hecho, una de las causas por la que se considera a Hopfield responsable de impulsar el desarrollo en el campo de las redes neuronales es precisamente el haber aplicado modelos matemáticos como éste, lo cual constituyó la base de posteriores trabajos sobre redes neuronales.

La función energía de una red Hopfield Discreta tiene la siguiente forma:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ i\neq j}}^{N} w_{ij}s_i s_j + \sum_{i=1}^{N}\theta_i s_i$$

Siendo:

w_{ij}: Peso de la conexión entre las neuronas i y j.

s_i : Valor de salida de la neurona i.

s_j : Valor de salida de la neurona j.

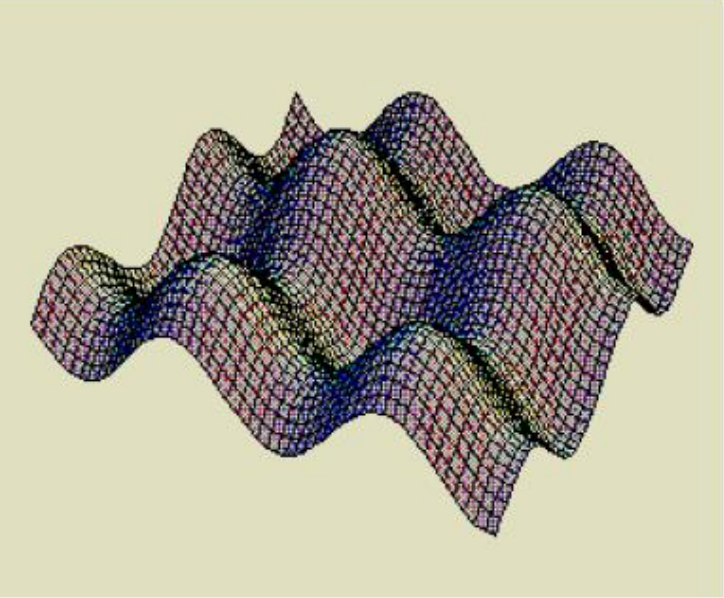
θ_i : Umbral de la función de activación de la neurona i.

Esta expresión guarda una profunda similitud formal con la energía mecánica clásica.

Trata de representar la evolución del sistema, considerando cada configuración (vector) de las salidas de las neuronas de la red como puntos en un espacio de dimensión N y relacionando el estado de la red en cada momento con un punto de ese espacio. La función energía puede imaginarse entonces como una superficie que presenta determinados valores mínimos, algo semejante a un paisaje montañoso donde los mínimos serían los valles (Figura 28). Cuando en la red se han almacenado M informaciones o patrones, los posibles estados estables de la red serán también M (durante su funcionamiento podrá responder ante una entrada con una salida que represente alguno de esos M patrones registrados). Estos M estados corresponden precisamente a los mínimos de la función energía. Cuando

se presenta a la entrada de la red una nueva información, ésta evoluciona hasta alcanzar un mínimo de la función energía, generando una salida estable.

Figura 28. Función energía de una red Hopfield



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.195.

Cuando la Red Hopfield se utiliza como memoria asociativa, el objetivo es conseguir que los patrones que debe memorizar se sitúen en los mínimos de la función y, consecuentemente, sean los estados estacionarios (estables) de la red. Puede demostrarse [Hopfield 82] que esto se consigue si se verifica que los pesos de las conexiones autorecurrentes son nulos (w_{ii}=0) y si el resto cumple la regla de Hebb (w_{ij}= $\sum s_i\, s_j$), de ahí que Hopfield eligiera dicho aprendizaje.

En cuanto a las redes de Hopfield Contínuas, su autor [Hopfield 84] considera que una posible expresión de su función de energía es la siguiente:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ i\neq j}}^{N} w_{ij}s_i s_j + \sum_{i=1}^{N}\int_0^{s_i} f^{-1}(s)ds$$

Donde f^{-1} es la inversa de la función de activación sigmoidal (f) de una neurona. Si la función f fuese, por ejemplo, $f(x) = \frac{1}{1+ e^{-\alpha(x-\theta_i)}}$,entonces la inversa seria f^{-1}(s)=$-(\frac{1}{2}\ln(-1+\frac{1}{s}))+\theta_i$ con lo que la función energía quedaría de la siguiente forma:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ i\neq j}}^{N} w_{ij}s_i s_j + \sum_{i=1}^{N}\theta_i s_i - \frac{1}{\alpha}\sum_{i=1}^{N}\int_0^{s_i} \ln(\frac{1}{s}-1)ds$$

Puede observarse que coincidiría con la función energía de la Red Hopfield Discreta si no existiera el último término, cuyo valor depende del parámetro α de la función de activación de las neuronas, que representa la ganancia o la pendiente de esta función sigmoidal. Por ello, habrá que tener especial cuidado en la elección de su valor, ya que influye directamente en los mínimos de la función energía. Si α tiene un valor ∞ (función de activación de tipo escalón), este término se hace cero y la red se convierte en una red Hopfield Discreta. Si el valor de α es finito pero muy grande (α >>1) el término puede despreciarse y los mínimos de la función de energía siguen siendo los mismos. Pero si α tiene valores pequeños, disminuye el número de mínimos de esta función y, como consecuencia, el número de posibles estados estables de la red, reduciéndose a un solo mínimo cuando α =0.

3.4.3 Limitaciones del modelo de Hopfield. Existen varios problemas asociados a la red Hopfield que han sido descritos por diferentes autores. En este apartado se mostrarán los dos más importantes, los que se refieren a la cantidad limitada de

datos que se pueden almacenar y a la necesidad de que estos datos sean ortogonales entre sí.

En una red de Hopfield el número de informaciones que puede ser "aprendido" (almacenado) está severamente limitado. Si se almacenan demasiadas informaciones, durante su funcionamiento la red puede converger a valores de salida diferentes de los "aprendidos" durante la etapa de entrenamiento, con lo que la tarea de asociación entre la información presentada y alguna de las almacenadas se realiza incorrectamente, pues se está asociando dicha información de entrada a otra desconocida.

Puede demostrarse que esta situación no se produce nunca si el número de informaciones almacenadas es menor o igual que N/4lnN, siendo N el número de neuronas de la red. Sin embargo, este límite puede suavizarse si se permite la posibilidad de un mínimo error en la recuperación de las informaciones almacenadas, suficientemente pequeño para poder identificar dicha información sin ningún problema, en tal caso el número de informaciones que se pueden almacenar debe ser menor que el 13.8% del número de neuronas de la red (0.138N). Por tanto, la capacidad o cantidad de informaciones de que puede almacenar una red Hopfield de N neuronas es:

$$Capacidad = \begin{cases} 0.138 \cdot N & para\ una\ recuperacion\ suficientemente\ buena \\ \dfrac{N}{4\ln(N)} & para\ una\ recuperacion\ perfecta \end{cases}$$

Esta es una seria limitación de la Red Hopfield, puesto que para almacenar unas pocas informaciones se precisará una gran cantidad de neuronas y, por tanto, un número muy elevado de conexiones. Por ejemplo, tomando la limitación del 13.8%, en una red de 100 neuronas, que tendría 100x99=9900 conexiones, sólo se podrían almacenar 13 informaciones.

Una segunda limitación del modelo es que, a pesar de cumplirse el requisito anterior, no siempre se puede garantizar que la red realice una asociación correcta entre una entrada y una de las informaciones almacenadas. Si éstas últimas no son suficientemente diferentes entre sí, es decir, si no son ortogonales, puede ocurrir que cada una de ellas no represente un mínimo de la función energía, con la probabilidad de que se generen salidas diferentes a todas ellas. También puede ocurrir que ante una entrada que coincida con una de las informaciones "aprendidas" la red converja hacia una salida correspondiente a otra de las informaciones almacenadas que fuese muy parecida.

Este problema puede ser minimizado mediante determinados procedimientos de ortogonalización que garanticen una diferencia suficiente entre las informaciones que debe "aprender" la red. En definitiva, lo que se debe pretender siempre es que estas informaciones sean ortogonales, lo cual ocurre si se cumple que cada par de patrones de entrada difieren en, al menos N/2 componentes, siendo N el número total de componentes por patrón. Esta condición puede expresarse como:

$$\sum_{i=1}^{N} e_i^{(k)} e_i^{(m)} \leq 0 \qquad \forall\, k \neq m$$

Donde $e_i^{(m)}$ y $e_i^{(k)}$ son los valores binarios (+1 o -1) de los componentes i-ésimos de dos vectores (patrones) diferentes (m y k) a almacenar en la red. Esta condición de ortogonalidad que establece el número de componentes diferentes (también llamado distancia Hamming) de dos patrones sea al menos de la mitad del total (*0.5N*) puede ser relajada, estableciendo una distancia mínima del 30% del total para que sean *casi* ortogonales, garantizándose todavía un funcionamiento aceptable. En este caso, considerando valores binarios de -1 y +1, la expresión sería $\sum e_i^{(k)} e_i^{(m)} \leq 0.7N - 0.3N = 0.4N$, ya que se permite un 70% de componentes diferentes y un 30% de iguales.

Siguiendo con el ejemplo, se podría comprobar si los vectores allí utilizados son ortogonales:

$$E_1 = [1,1,-1,-1] \qquad\qquad E_2 = [-1,-1,1,1]$$

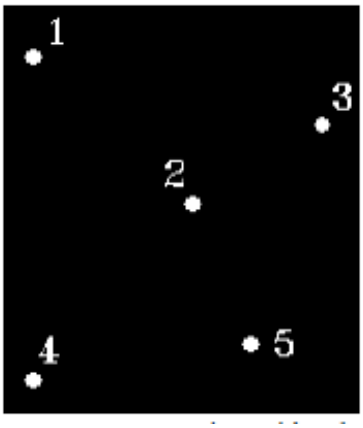
$$\sum_{i=1}^{4} e_i^{(1)} e_i^{(2)} = 1 \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 1 + (-1) \cdot 1 = -4 < 0$$

Por lo que E_1 y E2 son perfectamente ortogonales. En este caso, el número de componentes diferentes es el máximo posible, el 100% del total.

Algunos autores han propuesto modelos de redes autoasociativas que podrían ser más eficientes que el modelo de Hopfield. Por ejemplo, J. Anderson desarrolló la red conocida como Brain-State-In-A-Box. Se trata de una red con la misma topología que la anterior, pero con conexiones autorrecurrentes en todas las neuronas. En la red BSB también las informaciones que se almacenan deben ser ortogonales. Sin embargo, esta red tiene una mayor capacidad de almacenamiento ya que puede "aprender" un número de informaciones igual al número de neuronas que componen la red. Por ejemplo, una red BSB de 100 neuronas puede almacenar un máximo de 100 informaciones para que funcione correctamente (frente a las 13 de la Red Hopfield).

3.4.5.1 El problema del vendedor viajero. Este problema consiste en, dadas N ciudades que tiene que visitar un vendedor, encontrar el camino más corto para, partiendo de una de ellas, visitarlas toda sin pasar más de una vez por cada una y volviendo finalmente a la ciudad de partida. La complejidad reside en la enorme cantidad de posibles caminos, muchos de ellos de longitud similar. Para N ciudades el número de rutas alternativas es de *N!/2N*. De esta forma, para 5 ciudades existen 12 caminos posibles, para 10 el número de rutas es de 181.440 y para 50 del orden de $3x10^{62}$. Por ejemplo, podría considerarse el caso particular de 5 ciudades situadas como se indica en la Figura 29, con unas distancias entre ellas (en km) como las indicadas en la tabla 1.

Figura 29. Situación de las 5 ciudades del ejemplo



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.203.

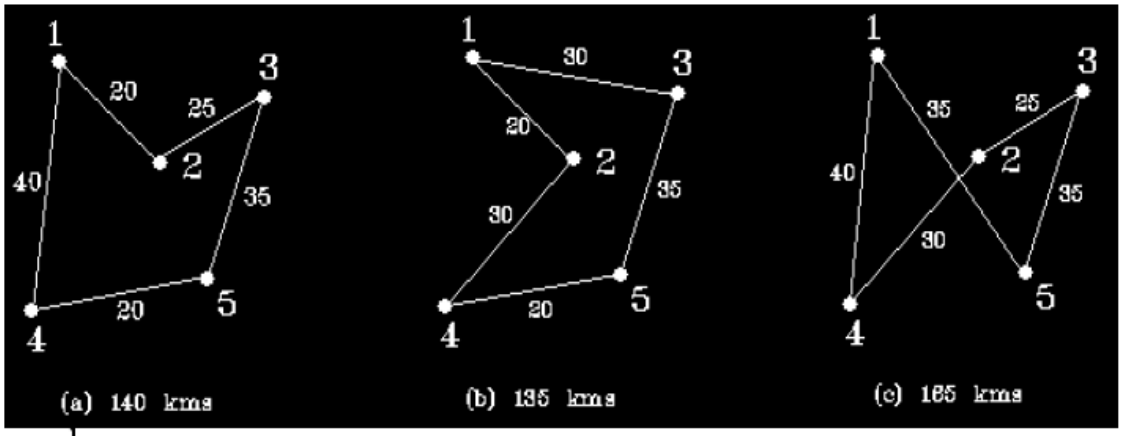
Tabla 1. Distancia (en km) entre las 5 ciudades del ejemplo

	1	2	3	4	5
1	0	20	30	40	35
2	20	0	25	30	15
3	30	25	0	50	35
4	40	30	50	0	20
5	35	15	35	20	0

Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.203.

En este ejemplo existirían 5!/2x5=12 rutas cerradas diferentes, cuya longitud dependerá del orden en que se visiten las ciudades. Así, en la figura 30 se indican tres de esas 12 posibles rutas.

Figura 30. Tres de las 12 posibles rutas alternativas con 5 ciudades



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.203.

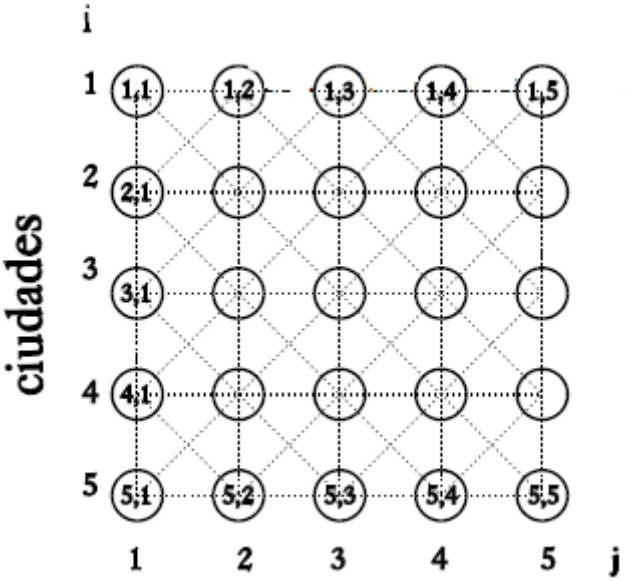
Este problema puede abordarse, en términos de la función energía, utilizando una red Hopfield de N^2 neuronas, siendo N el número de ciudades. Aunque la red usada en de tipo Contínua con valores en el margen [0,1], cuando se alcance una situación de estabilidad las salidas de las neuronas de la red tendrán valores binarios 0 (neurona activa) ó 1 (neurona inactiva), para lo cual se deberán tomar valores grandes para las ganancias o pendientes de las funciones de activación sigmoidales (tangentes hiperbólicas) de las neuronas. Entonces, cada neurona activa de la red (salida a 1) informaría del orden de una ciudad determinada en el recorrido del viajero. Para ello, hay que imaginarse las neuronas de la red como si estuvieran dispuestas en forma de matriz de N filas y N columnas (Figura 31), de tal forma que cada fila se asocia con una ciudad del recorrido y las columnas con la posición de cada ciudad dentro de la ruta. Así, si la neurona de la fila i y columna j está activada (s_{ij}=1) querrá decir que la ciudad i será la parada número j de la ruta que debería seguir el viajante.

Para poder resolver el problema, primero es necesario buscar una expresión de la función objetivo que se pretende minimizar, que puede tener el siguiente aspecto:

$$F = \frac{A}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{l=1\\ l\neq j}}^{N} s_{ij}s_{il} + \frac{B}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\ k\neq i}}^{N} s_{ij}s_{kj} + \frac{C}{2}(\sum_{i=1}^{N}\sum_{j=1}^{N} s_{ij} - N)^2 + \frac{D}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\ k\neq i}}^{N} d_{ij}(s_{ij}s_{kj+1} + s_{ij}s_{kj-1})$$

Donde i y k representan ciudades (filas de la matriz de neuronas), j y l representan paradas dentro de la ruta (columnas), d_{ij} representa la distancia entre las ciudades i y j, y A, B, C y D son constantes de proporcionalidad que sirven para cuantificar la importancia relativa de cada componente de la función objetivo.

Figura 31. Red Hopfield para resolver el problema del vendedor viaje



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.205.

Esta función se construye mediante términos que representan diferentes restricciones que deben satisfacerse para conseguir un resultado óptimo del problema, lo cual ocurrirá cuando el valor de estos términos sea mínimo. El primer término (en el que aparece la constante A) representa la restricción de que una

ciudad i sólo puede aparecer en una parada de la ruta, es decir, hay que evitar que se activen dos neuronas de la misma fila. Si ello se cumple, se anula por completo este término, puesto que, al multiplicar dos salidas de neuronas de la misma fila, el resultado será cero por no poder estar dos activas (salidas con valor "1") al mismo tiempo. El segundo término (en el que aparece B) representa la restricción de que una parada l de la ruta sólo puede corresponder a una ciudad. El tercero (C) obliga a que haya exactamente N ciudades en la ruta. Y el último (D) recoge la condición de que la distancia total del recorrido sea mínima.

Una vez establecida la función objetivo habrá que relacionarla con la expresión general de la función Energía de una red Hopfield:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} w_{ij,kl}s_{ij}s_{kl} + \sum_{i=1}^{N}\sum_{j=1}^{N}\theta_{ij}s_{ij}$$

Al enfrentar ambas funciones se obtiene que, para que sean equivalentes, los valores de los pesos de las conexiones entre dos neuronas, situadas una en la fila i y columna j, y otra en fila k y columna l, han de ser:

$$w_{ij,kl} = -A\delta_{ik}\left(1-\delta_{jl}\right) - B\delta_{jl}(1-\delta_{ik}) - C - Dd_{ik}(\delta_{j,l+1} + \delta_{j,l-1})$$

Donde δ_{xy} es la función delta de Kronecker cuyo valor es 1 si x=y, y 0 en caso Contrario.

La expresión para los umbrales de las funciones de activación de cada neurona en fila i y columna j será:

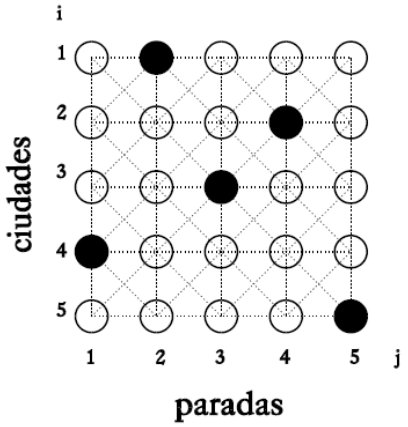
$$\theta_{ij} = -C \cdot N$$

Por tanto, si se implementa una Red Hopfield Contínua de NxN neuronas con unos umbrales y unos pesos con los valores anteriores y se hace iterar dicha red, se llegará a una situación de estabilidad en la que sólo habrá activas (salidas a "1") N

neuronas, una por fila y columna, que representarán el camino óptimo a seguir por el viajante. El principal problema de este método es el encontrar unos valores adecuados de las constantes A,B,C y D y de las ganancias de las funciones de activación sigmoidales de las neuronas para que la red converja hacia una solución óptima. En el caso de la ganancia, es posible que deba variar a lo largo del funcionamiento de la red, comenzando con valores bajos que se irán incrementando hasta alcanzar un valor alto que consiga salidas "0" y "1" en todas las neuronas.

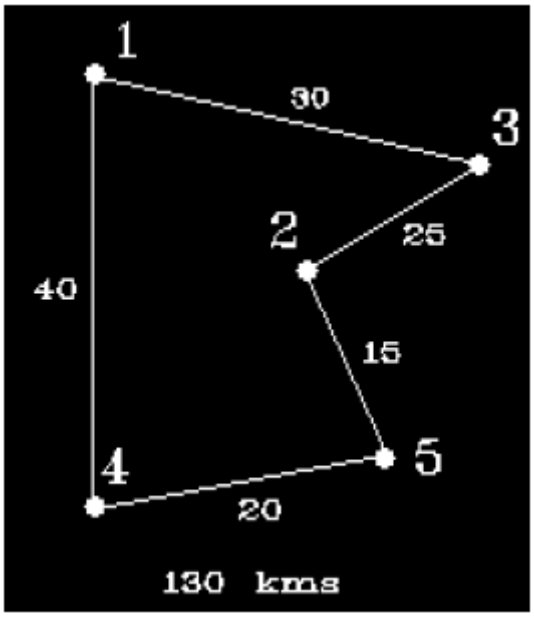
En el ejemplo con 5 ciudades presentado en la figura 29, si se implementa una Red Hopfield de 5x5=25 neuronas con los valores adecuados para los pesos y los umbrales y se pone en funcionamiento, se llega a una situación de estabilidad en la que sólo permanecen activadas las neuronas indicadas en la figura 44, con lo que la ruta óptima es la señalada en la figura 33, cuya longitud total es de 130 kms.

Figura 32. Situación final de la red del ejemplo. En negro las neuronas activadas (salida 1)



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales ArtificialeS. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.207.

Figura 33. Ruta óptima



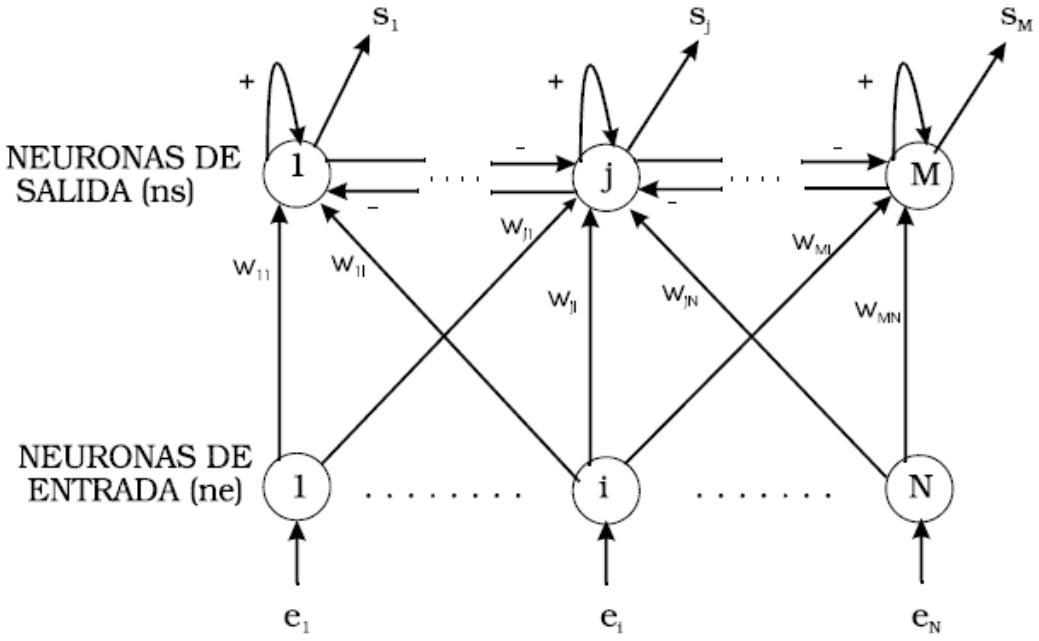
Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.207.

3.5 EL MODELO DE KOHONEN

Existen evidencias que demuestran que en el cerebro hay neuronas que se organizan en muchas zonas de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en forma de "mapas bidimensionales". Por ejemplo, en el sistema visual se han detectado "mapas" del espacio visual en zonas del córtex (capa externa del cerebro). También en el sistema auditivo se detecta una organización en base a la frecuencia a la que cada neurona alcanza la mayor respuesta (organización tonotópica).

3.5.1 Arquitectura. La arquitectura de la versión original (LVQ) del modelo de Kohonen es parecida a la de la red ART, aunque en este caso no existen conexiones feedback (Figura 34). Se trata de una red de dos capas con N neuronas de entrada y M de salida. Cada una de las N neuronas de entrada se conecta a las M de salida a través de conexiones hacia delante (feedfoward).

Figura 34. Arquitectura de la red LVQ (Learning Vector Quantization) de Kohonen

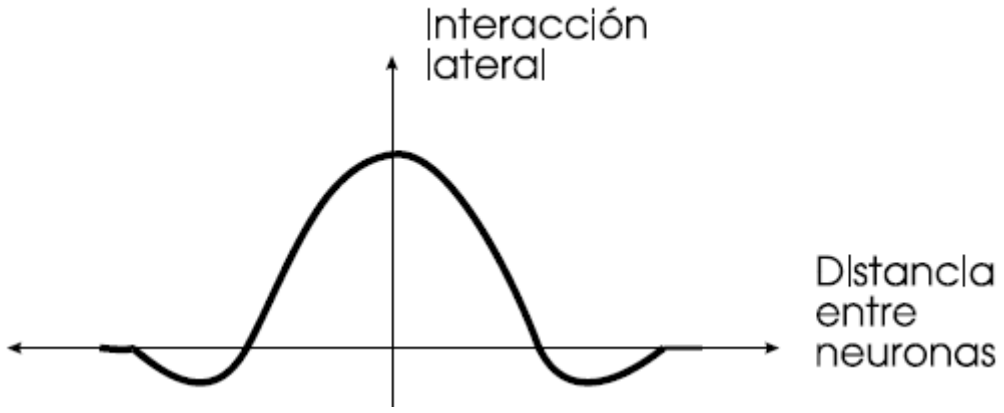


Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.254.

Entre las neuronas de la capa de salida puede decirse que existen conexiones laterales de inhibición (peso negativo) implícitas; pues aunque no estén conectadas, cada una de estas neuronas va a tener una cierta influencia sobre sus vecinas. El valor que se asigne a los pesos de las conexiones feedfoward entre las capas de entrada y salida (wji) durante el proceso de aprendizaje de la red va a depender precisamente de esta interacción lateral.

La influencia que una neurona ejerce sobre las demás es función de la distancia entre ellas, siendo muy pequeña cuando están muy alejadas. Es frecuente que dicha influencia tenga la forma de un "sombrero mejicano", como se ilustra en la figura 35. Esta afirmación tiene una base biológica, ya que existen evidencias fisiológicas de interconexiones laterales de este tipo entre las neuronas del sistema nervioso central de los animales. Así, se ha podido comprobar que en determinados primates se producen interacciones laterales de tipo excitatorio entre neuronas próximas en un radio de 50 a 100 micras, de tipo inhibitorio en una corona circular de 150 a 400 micras de anchura alrededor del círculo anterior, y de tipo excitatorio muy débil, prácticamente nulo, desde ese punto hasta una distancia de varios centímetros.

Figura 35. Interacción entre neuronas de la capa de salida

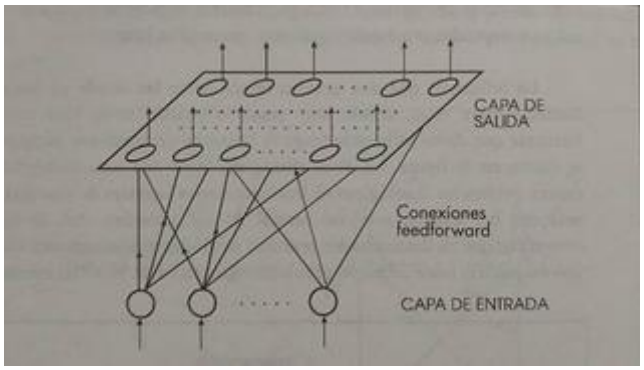


Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.254.

Por otra parte, la versión del modelo denominada TPM (*Topology Preserving Map*) trata de establecer una correspondencia entre los datos de entrada y un espacio bidimensional de salida, creando "mapas topológicos" de dos dimensiones, de tal forma que ante datos de entrada con características comunes se deben activar

neuronas situadas en zonas próximas de la capa de salida. Por esta razón, la representación habitual de esta red suele ser la mostrada en la figura 36, donde las M neuronas de salida se disponen de forma bidimensional para representar precisamente los mapas de características.

Figura 36. Arquitectura de la red TPM (Topology-Preserving Map) de Kohonen



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.256.

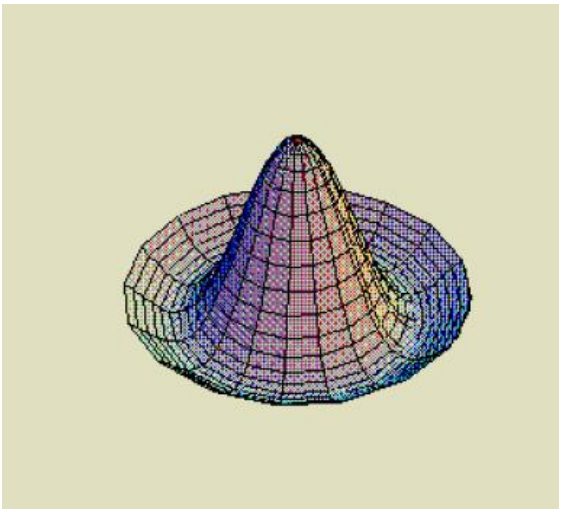
La interacción lateral entre las neuronas de la capa de salida sigue existiendo, aunque hay que entender la distancia ahora como una zona bidimensional que existe alrededor de cada neurona. Esta zona puede ser circular (Figura 37), cuadrada, hexagonal, o cualquier otro polígono regular centrado en dicha neurona.

3.5.2 Funcionamiento. El funcionamiento de esta red es relativamente simple. Cuando se presenta a la entrada una información $E_k = (e_1^{(k)}, \dots, e_N^{(k)})$, cada una de las M neuronas de la capa de salida la recibe a través de las conexiones feedfoward con pesos w_{ji}. También estas neuronas reciben las correspondientes entradas debidas a las conexiones laterales con el resto de neuronas de salida y cuya influencia dependerá de la distancia a la que se encuentren. Así, la salida generada por una neurona de salida j ante un vector de entrada E_k sería:

$$S_j(t+1) = f(\sum_{i=1}^{N} w_{ji} e_i^{(k)} + \sum_{p=1}^{M} Int_{pj} S_p(t))$$

Donde Int_{pj} es una función del tipo "sombrero mejicano" (Figuras 35 y 37) que representa la influencia lateral de la neurona p sobre la neurona j. La función de activación de las neuronas de salida (f) será del tipo contínuo, lineal o sigmoidal, ya que esta red trabaja con valores reales.

Figura 37. Interacción circular entre neuronas de la capa de salida



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.257.

Es evidente que se trata de una red de tipo competitivo, ya que, al presentar una entrada E_k la red evoluciona hasta una situación estable en la que se activa una neurona de salida, la vencedora. Por ello, la formulación matemática de su funcionamiento puede simplificarse mediante la siguiente expresión, que representa cuál de las M neuronas se activaría al introducir dicha información E_k:

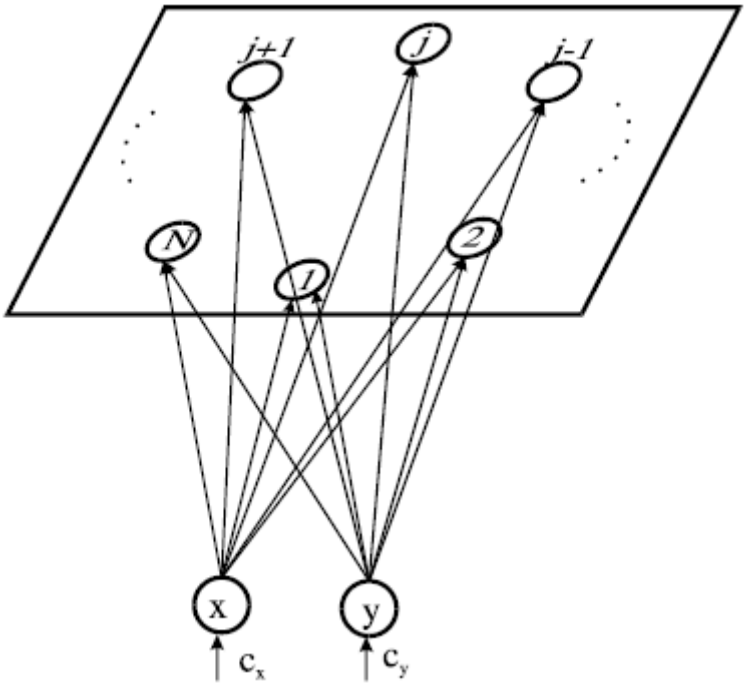
$$S_j = \begin{cases} 1 & MIN \left\| E_k - W_j \right\| = MIN \left(\sqrt{\sum_{i=1}^{N} (e_i^{(k)} - w_{ij})^2} \right) \\ 0 & resto \end{cases}$$

Donde $\left\| E_k - W_j \right\|$ es una medida (p. ej. distancia euclídea) de la diferencia entre el vector de entrada $E_k = (e_1^{(k)}, \dots, e_N^{(k)})$ y el vector de los pesos $W_j = (w_{j1}, \dots, w_{jN})$ de las conexiones entre cada una de las neuronas de entrada y la neurona de salida j. En estos pesos se registran los datos almacenados en la red durante el proceso de aprendizaje. En la fase de funcionamiento lo que se pretende es encontrar el dato "aprendido" más parecido al de entrada para, en consecuencia, averiguar qué neurona se activará y, sobre todo, en qué zona del espacio bidimensional de salida se encuentra.

Lo que hace la red de Kohonen, en definitiva, es realizar una tarea de clasificación, ya que la neurona de salida activada ante una entrada representa la clase a la que pertenece dicha información de entrada. Además, como ante otra entrada parecida se activa la misma neurona de salida, u otra cercana a la anterior, debido a la semejanza entre las clases, se garantiza que las neuronas topológicamente próximas sean sensibles a entradas físicamente similares. Por esta causa, la red es especialmente útil para establecer relaciones, desconocidas previamente, entre conjuntos de datos.

3.5.4.1 Resolución de problema del Agente Viajero. El modelo de Kohonen se ha utilizado también para intentar resolver el problema de optimización conocido como *problema del viajante* o del *vendedor viajero*, ya presentado en este capítulo, estableciendo el camino más corto entre una serie de ciudades sin pasar dos veces por la misma.

Figura 38. Red de Kohonen para resolver el problema del viajante



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.272.

La red que puede solucionar este problema tiene tantas neuronas de salida como ciudades del recorrido y dos neuronas de entrada para las coordenadas en el plano (x,y) de cada ciudad (Figura 39). La distribución de las neuronas de salida es circular, estableciendo para cada neurona (j) una zona de vecindad que incluye las neuronas anterior (j-1) y posterior (j+1).

El proceso a seguir para resolver el problema consiste en entrenar la red para que "aprenda" la situación geográfica de las N ciudades del recorrido, adaptando los pesos mediante el algoritmo de Kohonen. Cuando el entrenamiento ha concluido, al existir tantas neuronas de salida como ciudades, deberán coincidir los vectores de pesos de la red $W_j = [w_{jx}, w_{jy}]$ con las coordenadas de las N ciudades ($C_1 = [c_x^{(1)}, c_y^{(1)}]$,…, $C_N = [c_x^{(N)}, c_y^{(N)}]$) Entonces el problema ha sido resuelto, siendo la

solución el recorrido resultante de unir los puntos (ciudades) correspondientes a los pesos de las neuronas adyacentes de la capa de salida. La solución que se obtiene es buena, aunque no puede garantizarse que sea la óptima, ya que el criterio seguido por la red, al fijar la zona de vecindad indicada, ha sido el buscar siempre como siguiente ciudad la más próxima.

En este caso, en lugar de distancias, se trabajará con las coordenadas de las ciudades en el plano X-Y, cuyos valores son:

$$C_1=[1,9] \quad C_2=[6,5] \quad C_3=[9,7] \quad C_4=[2,1] \quad C_5=[7,2]$$

A partir de las coordenadas, es directo el cálculo de las distancias entre las ciudades. Suponiendo que se está trabajando con valores expresados en kilómetros, las distancias serían las indicadas en la tabla 2.

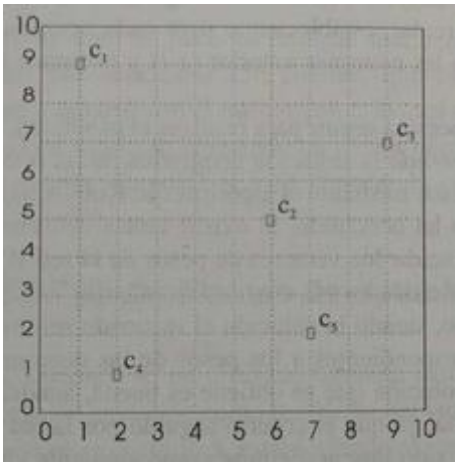
Tabla 2. Distancia (en km) entre las 5 ciudades del ejemplo

	C_1	C_2	C_3	C_4	C_5
C_1	0.00	6.40	8.25	8.06	9.22
C_2	6.40	0.00	3.60	5.66	3.16
C_3	8.25	3.60	0.00	9.22	5.38
C_4	8.06	5.66	9.22	0.00	5.10
C_5	9.22	3.16	5.38	5.10	0.00

Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.273.

Puede comprobarse que el recorrido más corto, corresponde con la secuencia C1-C3-C2-C5-C4-C1, con una distancia de valor: Distancia mínima= $d_{13}+d_{32}+d_{25}+d_{54}+d_{41}$ = 8.25+3.60+3.16+5.10+8.06 = 28.17 [km]

Figura 39. Situación de las 5 ciudades del ejemplo



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.274.

Se puede utilizar una red de Kohonen para obtener una solución del problema. En este caso se trataría de una red con 2 neuronas de entrada para las coordenadas X e Y de cada ciudad y 5 neuronas de salida para disponer de 5 vectores (W_1,..,W_5) de pesos $W_j = [w_{jx}, w_{jy}]$, uno por ciudad, ya que cuando la red "aprenda" las posiciones de las ciudades, el valor de los pesos coincidirá con dichas posiciones, siendo el orden del recorrido el de los pesos: primero se pasará por la ciudad almacenada en W1, después por la que indique W2 y finalmente por la de W5. Por tanto, la solución del problema consiste en averiguar el reparto de ciudades por pesos para conocer la secuencia del recorrido.

El algoritmo de aprendizaje es el ya conocido:

$$w_{jx}(t+1) = w_{jx}(t) + \alpha(t)\left[c_x^{(k)} - w_{j^*x}(t)\right] \quad para\ j = j^* - 1, j^*, j^* + 1$$

$$w_{jy}(t+1) = w_{jy}(t) + \alpha(t)\left[c_y^{(k)} - w_{j^*y}(t)\right] \quad para\ j = j^* - 1, j^*, j^* + 1$$

Siendo j^* la neurona vencedora al aplicar a la entrada las coordenadas de la ciudad k: $C_k = [c_x^{(k)}, c_y^{(k)}]$. En este caso, la zona de vecindad alrededor de la neurona vencedora j* contiene únicamente a las neuronas anterior y posterior a aquella; por tanto, se realiza el ajuste de los pesos Wj*-1, Wj*, y Wj*+1. En este ejemplo se ha tomado un coeficiente de aprendizaje proporcional a
1/ t(α (t)=0.5/t).

Inicialmente (t=0) se deben establecer unos valores aleatorios para los pesos. En nuestro caso, los valores elegidos son puntos de una hipotética circunferencia de radio 2 km (Figura 40). Estos valores aleatorios influirán en la solución final del problema. Ya se ha comentado que este método ofrece una solución buena, buscando siempre la siguiente ciudad más cercana a la anterior, pero no siempre óptima. En este ejemplo, con los valores iniciales asignados a los pesos se llega precisamente a la solución óptima.
Aplicando el algoritmo anterior se obtiene la adaptación de pesos indicada en la tabla 3.

Tabla 3. Evolución de los pesos

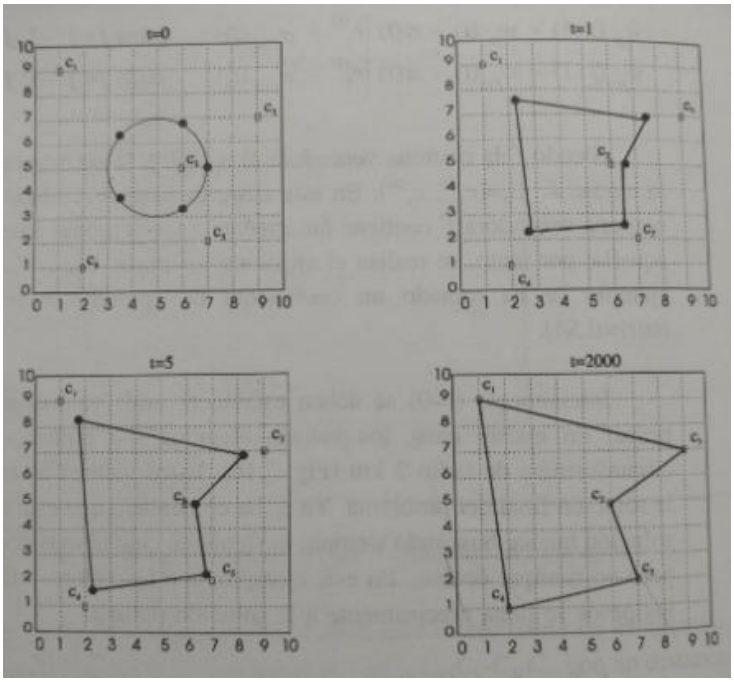
W_j	t=0	t=1	t=2	t=5	t=260	t=2000
W_1	[3.5,6.32]	[2.25,7.66]	[1.94,7.99]	[1.62,8.34]	[1.09,8.91]	[1.03,8.97]
W_2	[6.0,6.73]	[7.50,6.86]	[7.88,6.90]	[8.26,6.93]	[8.90,6.99]	[8.96,7.00]
W_3	[7.0,5.00]	[6.50,5.00]	[6.38,5.00]	[6.25,5.00]	[6.03,5.00]	[6.01,5.00]
W_4	[6.0,3.27]	[6.50,2.63]	[6.62,2.48]	[6.75,2.31]	[6.97,2.04]	[6.99,2.02]
W_5	[3.5,3.68]	[2.75,2.34]	[2.56,2.01]	[2.37,1.66]	[2.05,1.09]	[2.02,1.03]

$W_j = [w_{jx}, w_{jy}]$ Durante el aprendizaje de las posiciones de las 5 ciudades

Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.275.

Después de las primeras iteraciones ya puede observarse hacia qué ciudad tiende cada vector de pesos (Figura 40). Cuando se repite el proceso algunos cientos de veces, se llega a unos valores que prácticamente coinciden con las posiciones de las ciudades. En este ejemplo, después de 200 iteraciones coinciden los valores de las posiciones con una diferencia menor de una décima de kilómetro.

Figura 40. Adaptación de los pesos de una red de Kohonen



que soluciona el problema del viajante con 5 ciudades

Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.276.

Aunque no ocurre en este caso, cuando se resuelve el problema del viajante con una red de Kohonen puede suceder que más de una neurona pueda ser atraída por la misma ciudad, lo que originaría que algunas ciudades quedasen aisladas sin aparecer en el recorrido final. Esta situación puede evitarse utilizando un mayor número de neuronas que de ciudades.

4. OPTIMIZACIÓN DE TRAYECTORIA

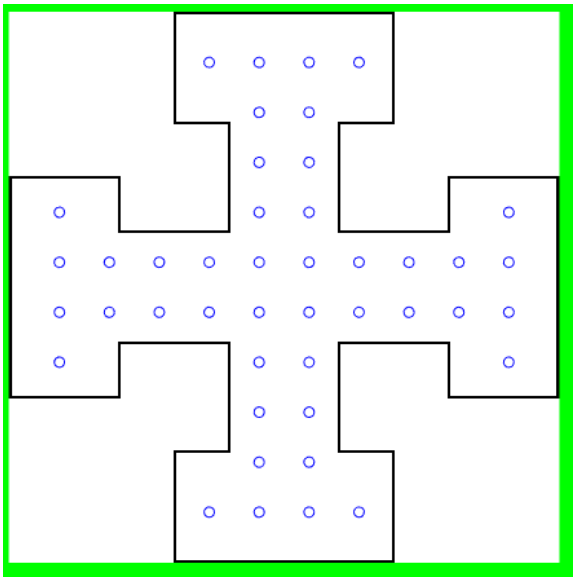
4.1 OPTIMIZACIÓN DE TRAYECTORIA POR PROGRAMACIÓN LINEAL.

Para la optimización del problema del agente viajero se toman los puntos maquinables y la función solucionará el problema inicial y se verá que la solución tiene subtures, esto significa que la solución óptima encontrada no da una ruta continua a través de todos los puntos, sino que tiene varios bucles desconectados, seguidamente se utilizará un proceso iterativo para determinar los sub-contadores, agregar restricciones y volver a ejecutar la optimización hasta que se eliminen los sub-contadores.

El proceso que realiza el programa para la optimización es el siguiente.

- Tomar todos los puntos a los cuales va a llegar (pares de paradas), en este paso se toman los puntos a mecanizar definidos en el step 2 y se asignan sus posiciones en 2 vectores cada uno con las coordenadas tanto en x como en y.

Figura 41. Puntos a mecanizar



- Calcular la distancia de cada viaje posible de un punto a otro. En este paso se toman las distancias gracias a las reglas pitagóricas aplicándolas entre todos los puntos entre sí.

```
dist = hypot(stopsLat(idxs(:,1)) - stopsLat(idxs(:,2)), ...
        stopsLon(idxs(:,1)) - stopsLon(idxs(:,2)));
lendist = length(dist);
```

- Las variables de decisión son binarias y asociadas a cada viaje, donde cada 1 representa un viaje que existe en el recorrido, y cada 0 representa un viaje que no está en el recorrido, esto se realiza tomando un limite superior de 1 y uno inferior de 0.

```
intcon = 1:lendist;
lb = zeros(lendist,1);
ub = ones(lendist,1);
```

- Designación de límites de igualdad.
  La primera restricción es que debe haber igual número de viajes como número de puntos a visitar.

```
Aeq = spones(1:length(idxs));
beq = nStops;
```

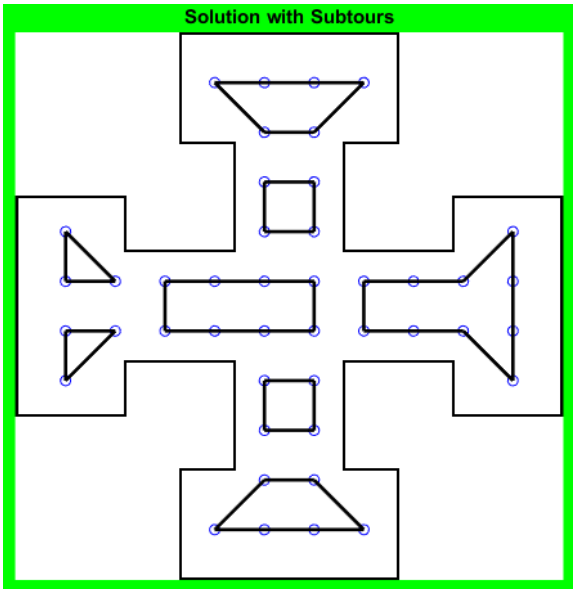
La segunda restricción es garantizar que el recorrido incluya todos los puntos, para ello se dice que cada punto está en exactamente dos viajes. Esto significa una llegada al punto y una salida del mismo punto

```
Aeq = [Aeq;spalloc(nStops,length(idxs),nStops*(nStops-1))];
for ii = 1:nStops
   whichIdxs = (idxs == ii);
   whichIdxs = sparse(sum(whichIdxs,2));
```

```
end
    Aeq(ii+1,:) = whichIdxs';
end
beq = [beq; 2*ones(nStops,1)];
```

Una vez se tiene las anteriores definiciones listas se resuelve el problema de la optimización usando la función intlinprog, la cual es una función propia del Toolbox de MATLAB la cual exige como entradas las distancias, los limites o restricciones, esta función devuelve una solución óptima dividida en otras más cortas como se puede observar en la figura 42

Figura 42. Iteración de Programación Lineal



Seguidamente se selecciona uno de los subtures y se buscan las restricciones para esa ruta, se crean las desigualdades para prohibir ese subtour y todo el subtour que usa esas paradas.

Se vuelve a usar la función intlinprog, adicionando a los parámetros de entrada las nuevas restricciones, se obtiene una nueva solución con un nuevo número de subtures, se repite lo anterior hasta que encuentra una única ruta

Figura 43. Proceso de identificación de subtures

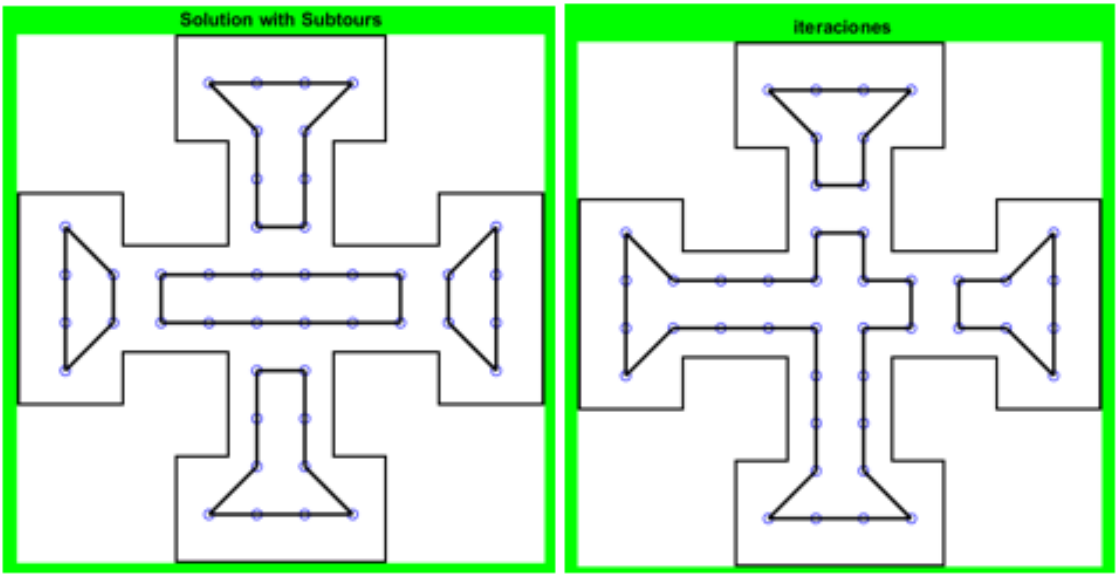
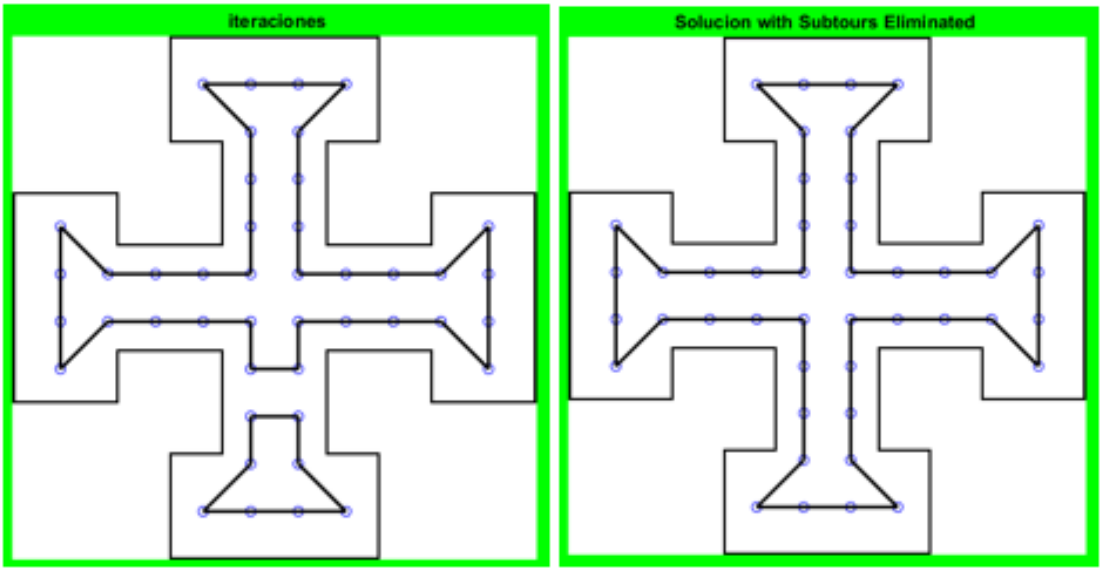


Figura 44. Optimización de subtures



Una vez se encuentra una única trayectoria se actualizan y guardan los puntos en la tabla 4 en el orden que deben ser maquinados

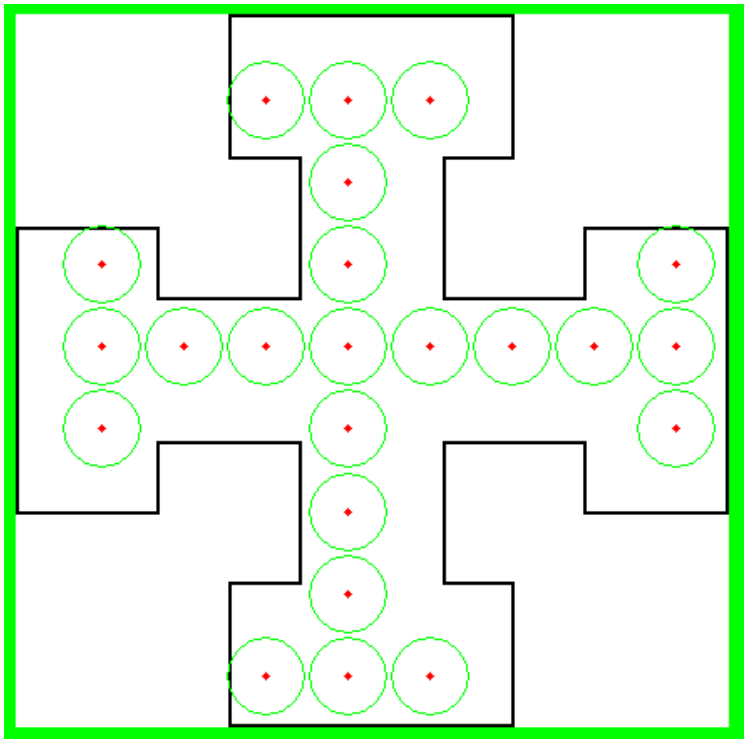
Tabla 4. Puntos de Maquinado

	X	Y
1	4.7244	0.9652
2	5.6642	0.9652
3	6.6040	0.9652
4	5.6642	1.9050
5	5.6642	2.8448
6	5.6642	3.7846
7	5.6642	4.7244

4.2 OPTIMIZACIÓN DE TRAYECTORIA POR MODELO DE HOPFIELD.

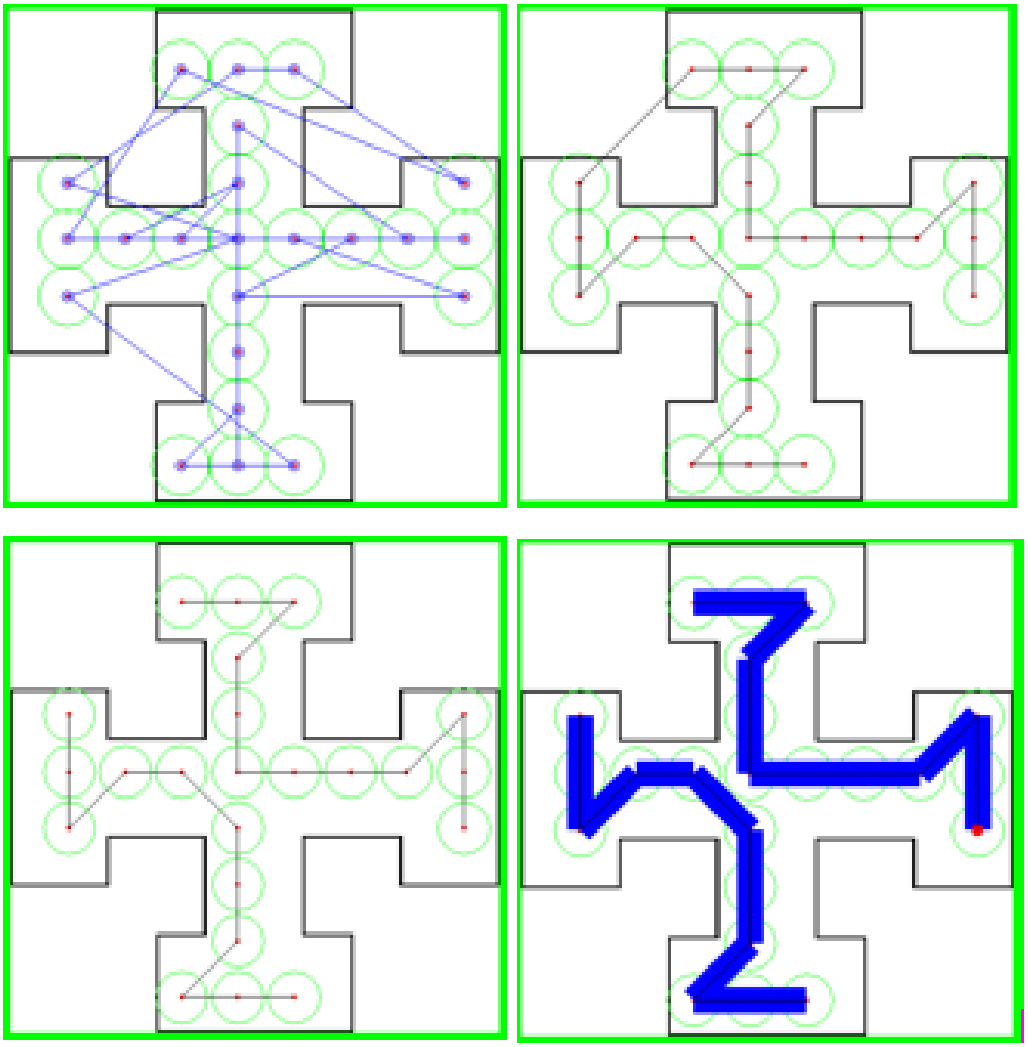
Al momento de iniciar este proceso hay una ventana con el mapa de la cavidad y los puntos admisibles para el maquinado en los botones de optimizar se seleccionan “optimización Hopfield”.

Figura 45. Puntos maquinables



Para trazar una trayectoria óptima el programa debe resolver el problema del TSP (travelling salesman problem), a través del método de mapas autoorganizables conformando un anillo o banda elástica que a través de una red neuronal encuentra la trayectoria óptima para ese conjunto de puntos. La trayectoria óptima es la distancia mínima en la que la herramienta pasa por todos los puntos sin devolverse (Ver figura 46).

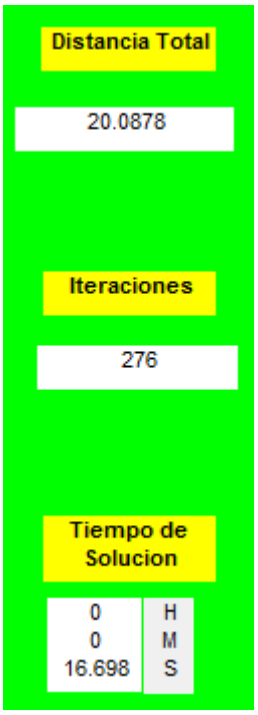
Figura 46. Proceso de optimización Hopfield



Una vez calculados y ubicados los datos iniciales, se inicia el proceso de aprendizaje en el cual la función o el modelo de Hopfield busca optimizar la función de energía

El software determina el número de iteraciones que debe realizar, el tiempo de optimización y la distancia recorrida, este se detiene automáticamente al completar el proceso (ver figura 47).

Figura 47. Determinación de Distancia, iteraciones y tiempo



Finalizado el proceso de Optimizar la trayectoria es necesario corregir ciertas rutas que son indeseables en la trayectoria trazada (Ver figura 48).

Figura 48. Corrección de rutas indeseadas

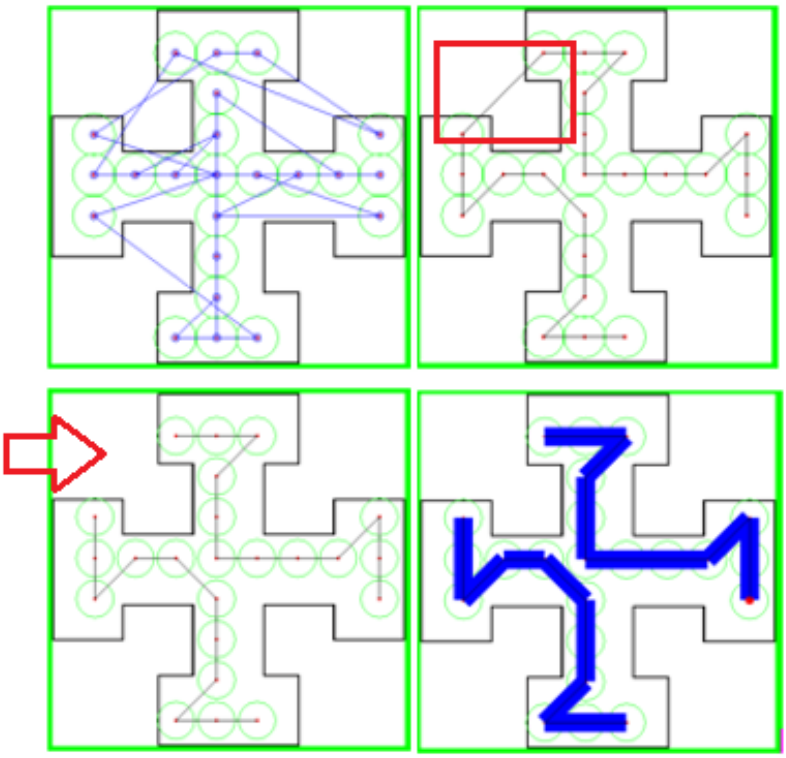


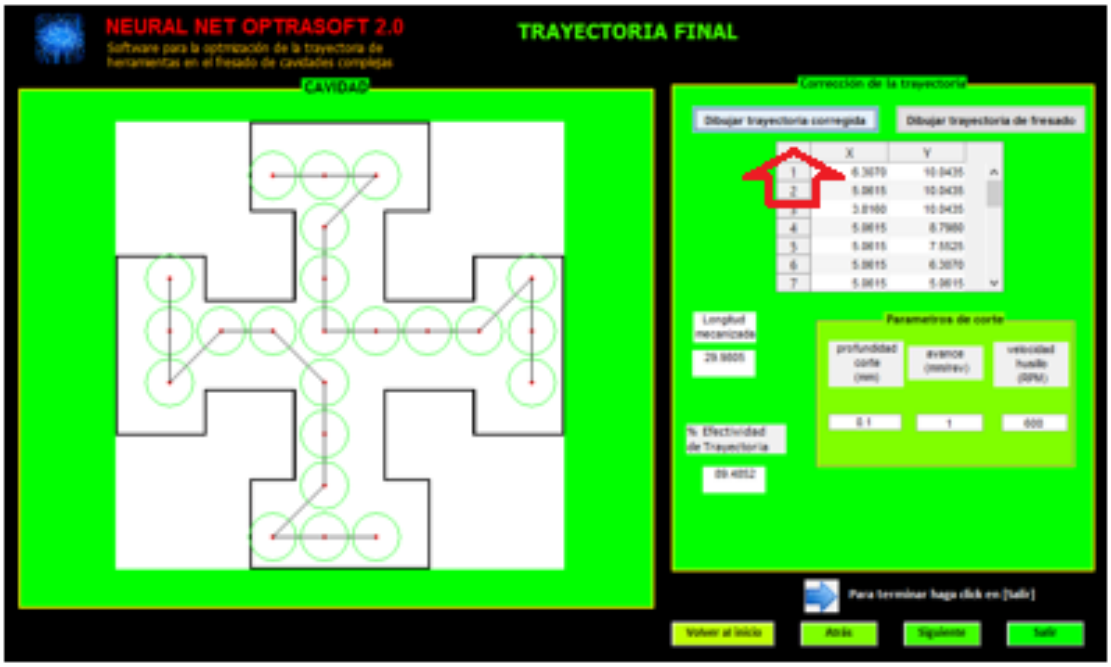
Figura 49. Coordenadas corregidas

	X	Y
1	6.3070	10.0435
2	5.0615	10.0435
3	3.8160	10.0435
4	5.0615	8.7980
5	5.0615	7.5525
6	5.0615	6.3070
7	5.0615	5.0615

Pasos finales. Corrección de la trayectoria (Ver figura 50).

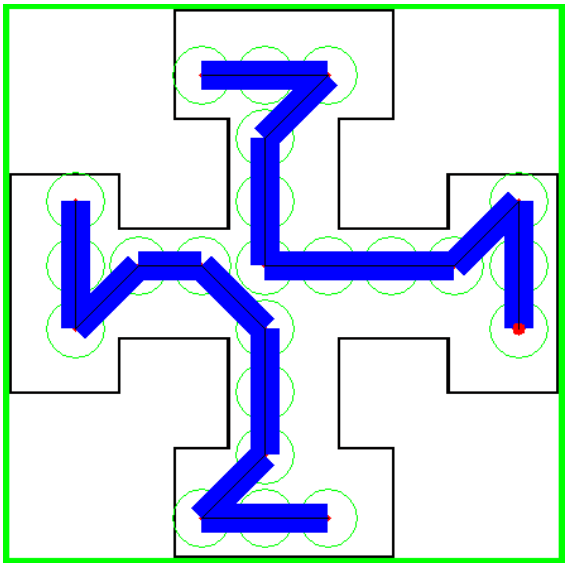
Haga clic en el botón “Dibujar trayectoria corregida” para dibujar la trayectoria corrigiendo los saltos de la trayectoria o los cruces (Ver figura 51), es importante resaltar que debido a que el software corrige los saltos, la distancia de la optimización también se verá corregida, por tanto en el step se puede visualizar la distancia real y el porcentaje de corrección de trayectoria

Figura 50. Corrección de la trayectoria



A continuación se hace clic en el botón “Dibujar trayectoria de fresado” donde se puede observar el proceso de fresado cuya finalidad es mostrar la zona desbastada. Hasta este paso concluye el proceso de obtención de la trayectoria (Ver Figura 51).

Figura 51. Trayectoria de fresado



Luego de generar la trayectoria de fresado, se pueden introducir algunos valores en un recuadro para modificar los parámetros de corte (la profundidad de corte, el avance y la velocidad del husillo) (ver Figura 52) Finalmente el software mostrará utilizando el parámetro de profundidad de la cavidad, un modelo de la pieza y la trayectoria de fresado (Ver figura 53) y las coordenadas de los puntos (ver figura 54).

Figura 52. Parámetros de corte

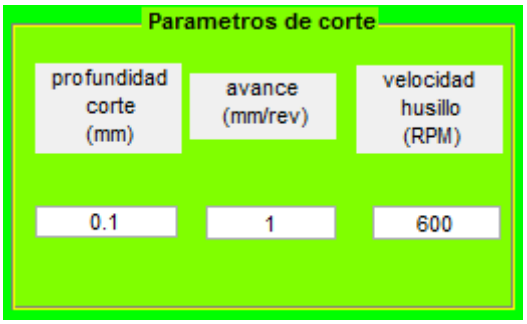


Figura 53. Modelo de la pieza

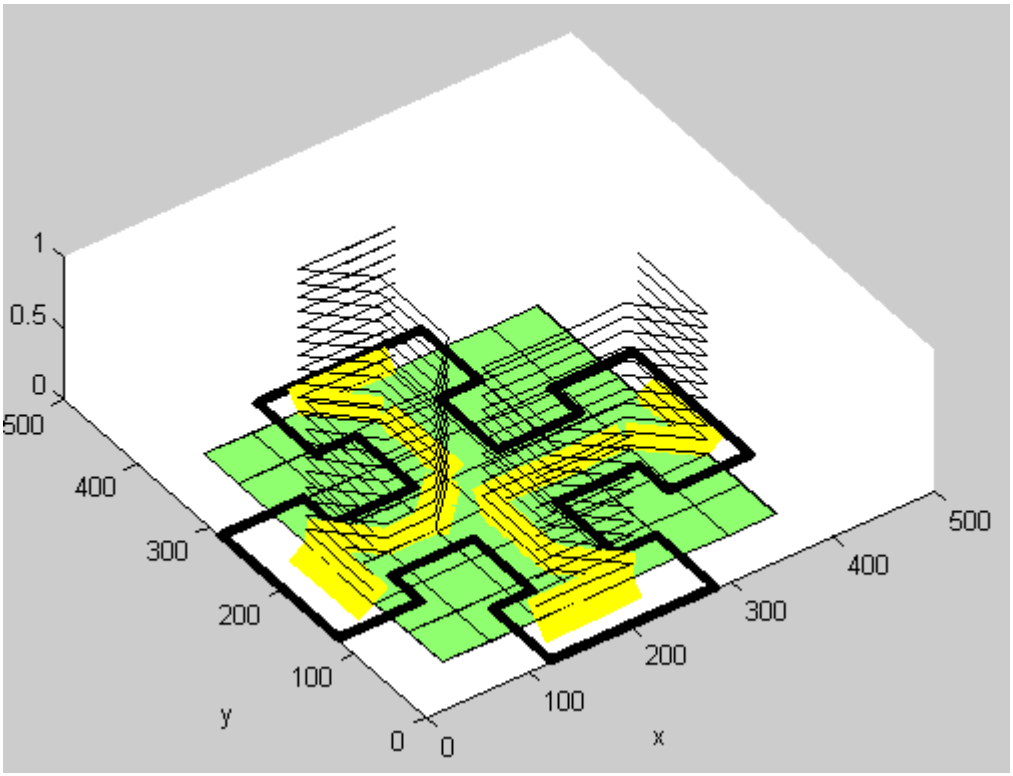


Figura 54. Coordenadas de los puntos

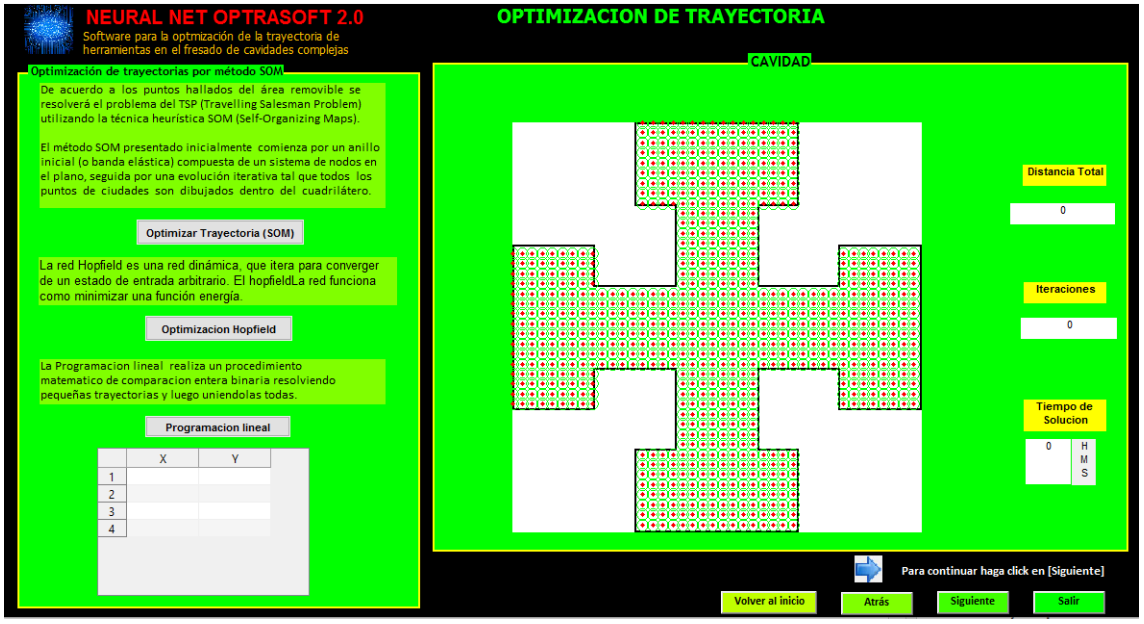
Archivo Edición Formato Ver Ayuda

238.00	191.00
144.00	191.00
191.00	191.00
191.00	144.00
97.00	50.00
50.00	50.00
144.00	191.00
238.00	191.00
191.00	238.00
285.00	332.00

4.3 OPTIMIZACIÓN DE TRAYECTORIA POR MODELO DE KOHONEN.

Al momento de iniciar este proceso hay una ventana con el mapa de la cavidad y los puntos admisibles para el maquinado en los botones de optimizar se selecciona “optimizar trayectoria SOM” (Ver figura 55).

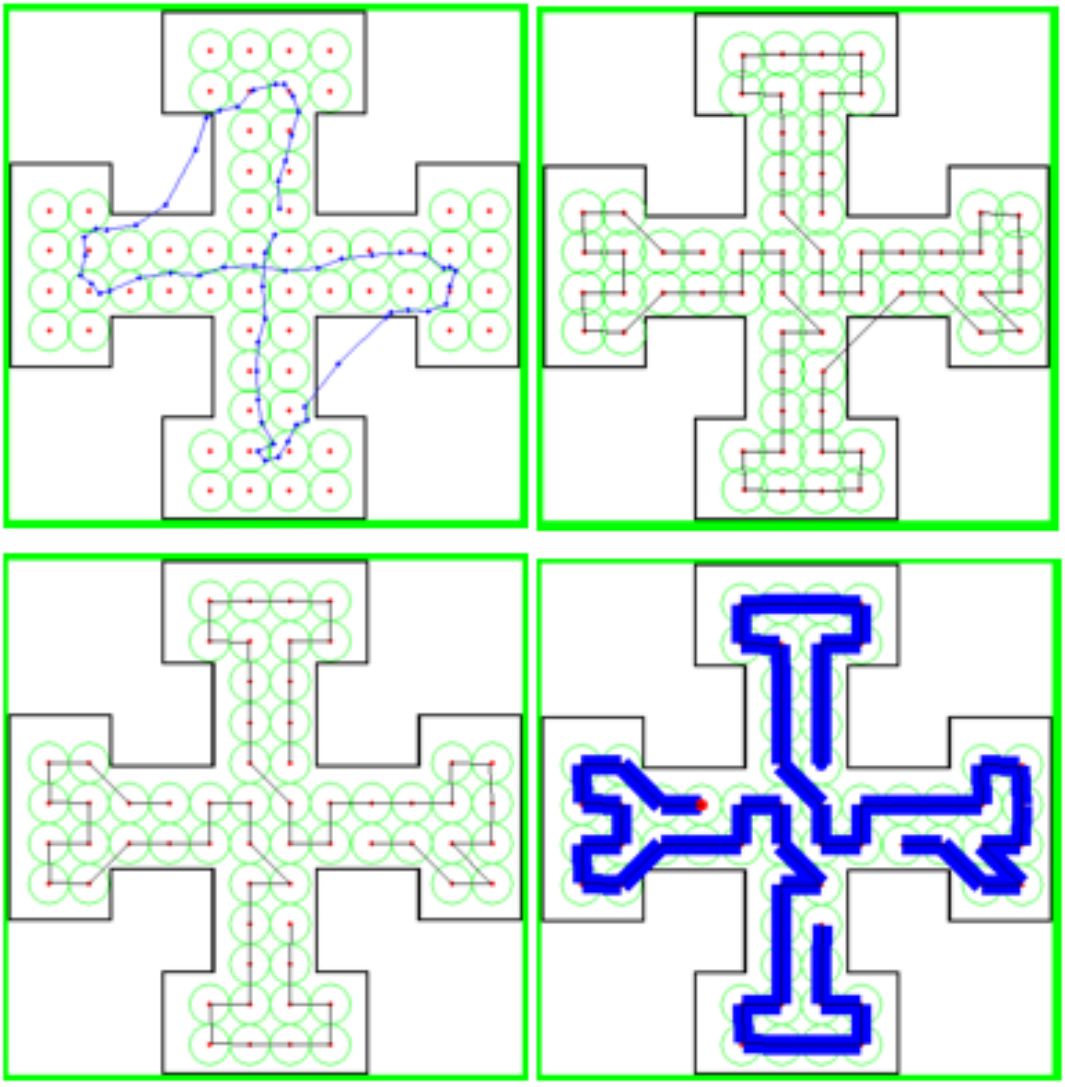
Figura 55. Puntos maquinables



Para trazar una trayectoria óptima el programa debe resolver el problema del TSP (travelling salesman problem), a través del método de mapas autoorganizables conformando un anillo o banda elástica que a través de una red neuronal encuentra

la trayectoria óptima para ese conjunto de puntos. La trayectoria óptima es la distancia mínima en la que la herramienta pasa por todos los puntos sin devolverse (Ver figura 56).

Figura 56. Proceso de optimización trayectoria SOM

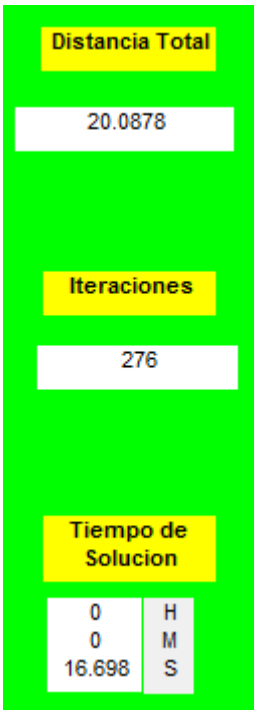


Una vez calculados y ubicados los datos iniciales, se inicia el proceso de aprendizaje off-line (no supervisado), haciendo clic en el botón “Optimizar

trayectoria SOM", se inicia el proceso de funcionamiento de la red descrito anteriormente.

El software determina el número de iteraciones que debe realizar, el tiempo de optimización y la distancia recorrida, este se detiene automáticamente al completar el proceso (ver figura 57).

Figura 57. Determinación de Distancia, iteraciones y tiempo



Finalizado el proceso de Optimizar la trayectoria es necesario corregir ciertas rutas que son indeseables en la trayectoria trazada (Ver figura 58).

Figura 58. Corrección de rutas indeseadas

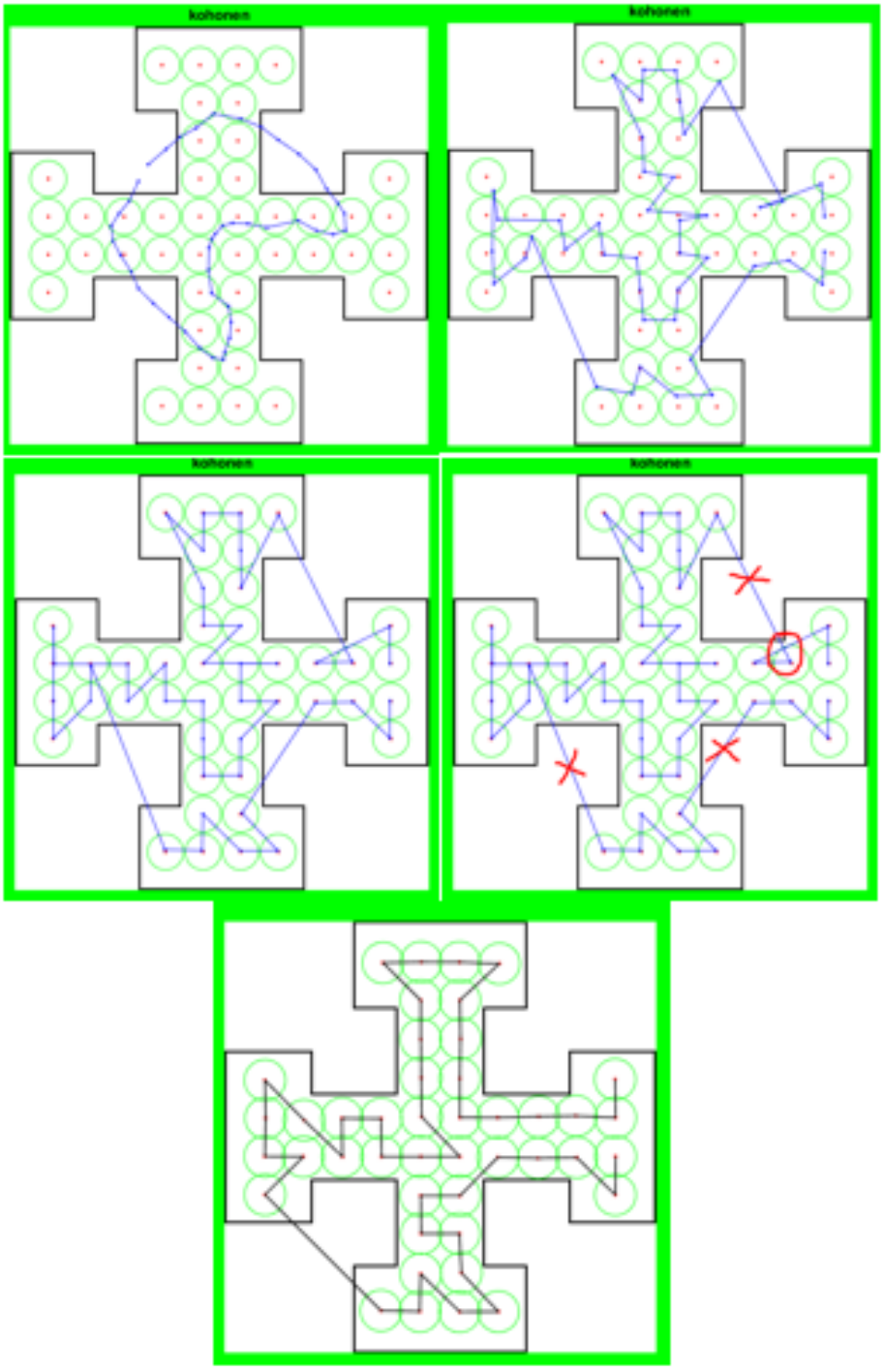


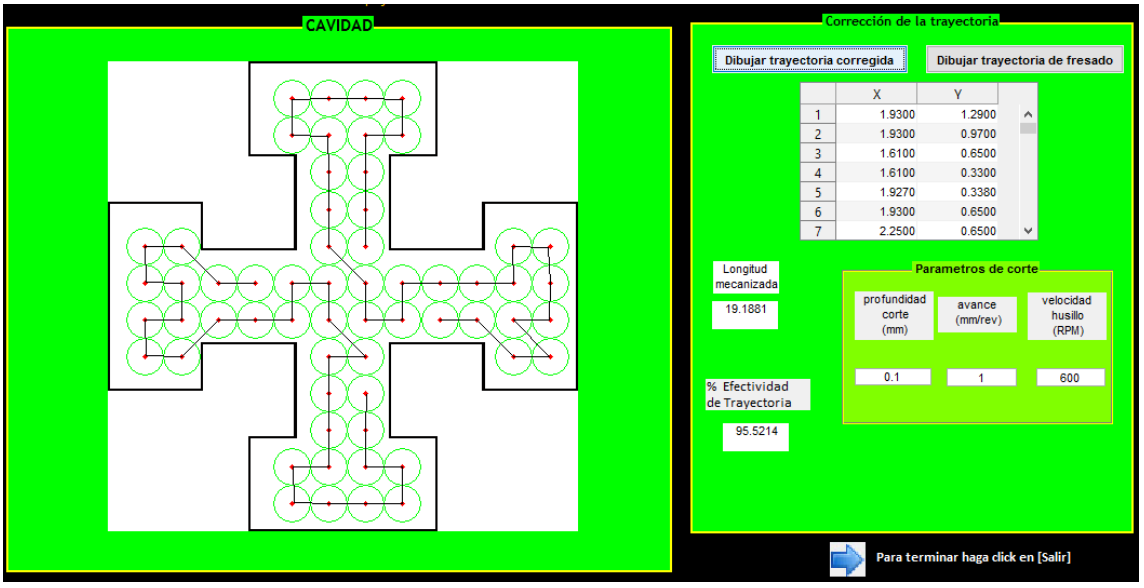
Figura 59. Coordenadas corregidas

	X	Y
1	1.9300	1.2900
2	1.9300	0.9700
3	1.6100	0.6500
4	1.6100	0.3300
5	1.9270	0.3380
6	1.9300	0.6500
7	2.2500	0.6500

Pasos finales. Corrección de la trayectoria (Ver figura 60).

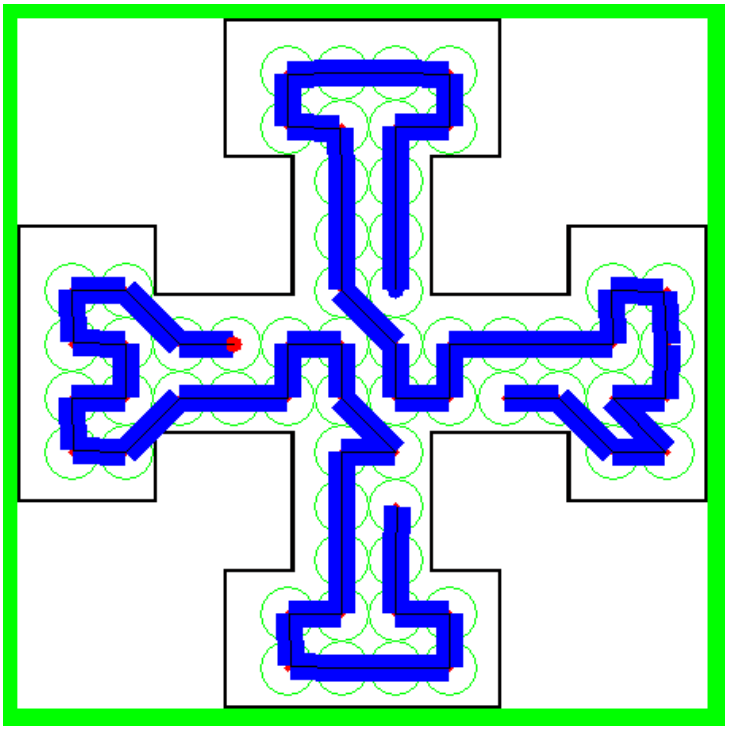
Haga clic en el botón “Dibujar trayectoria corregida” para dibujar la trayectoria corrigiendo los saltos de la trayectoria o los cruces (Ver figura 60), es importante resaltar que debido a que el software corrige los saltos, la distancia de la optimización también se verá corregida, por tanto en el step se puede visualizar la distancia real y el porcentaje de corrección de trayectoria

Figura 60. Corrección de la trayectoria



A continuación se hace clic en el botón “Dibujar trayectoria de fresado” donde se puede observar el proceso de fresado cuya finalidad es mostrar la zona desbastada. Hasta este paso concluye el proceso de obtención de la trayectoria (Ver Figura 61).

Figura 61. Trayectoria de fresado



Luego de generar la trayectoria de fresado, se pueden introducir algunos valores en un recuadro para modificar los parámetros de corte (la profundidad de corte, el avance y la velocidad del husillo) (ver Figura 62) Finalmente el software mostrará utilizando el parámetro de profundidad de la cavidad, un modelo de la pieza y la trayectoria de fresado (Ver figura 63) y las coordenadas de los puntos (ver figura 64).

Figura 62. Parámetros de corte

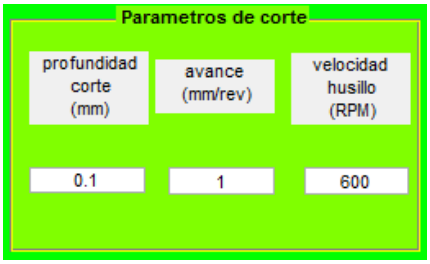


Figura 63. Modelo de la pieza

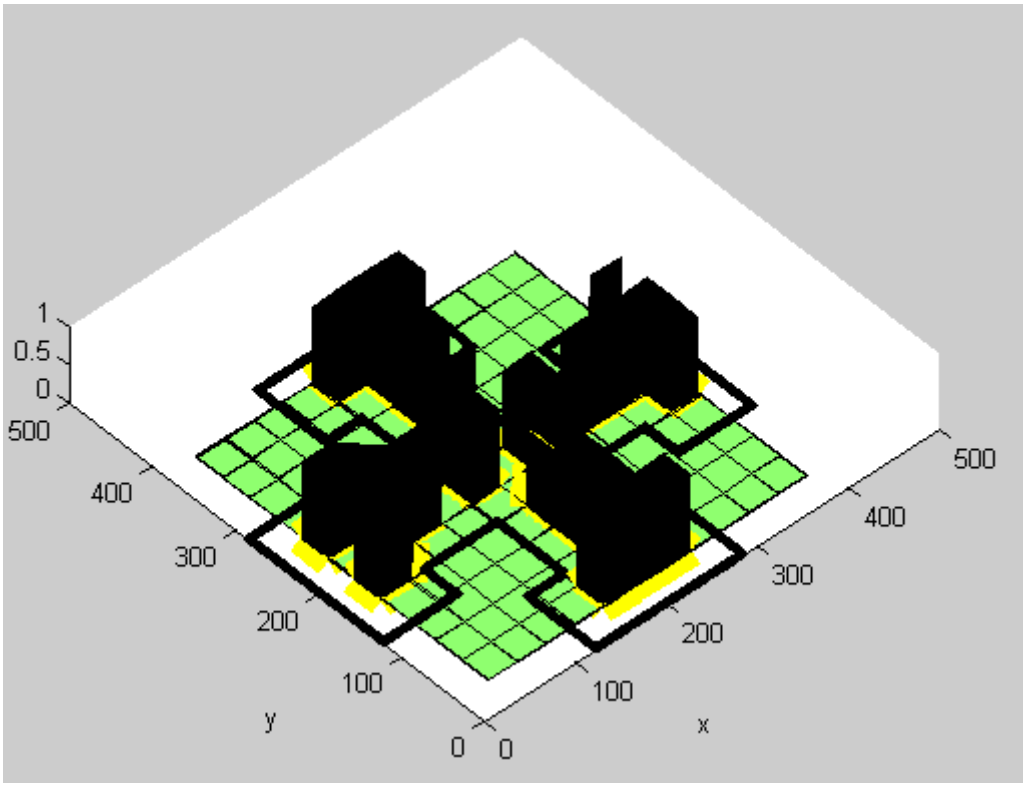


Figura 64. Coordenadas de los puntos

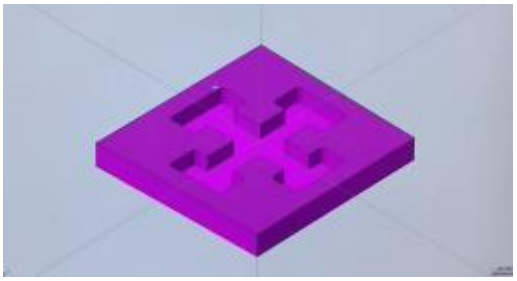
Archivo Edición Formato Ver Ayuda

193.00	193.00
161.00	161.00
192.70	193.00
225.00	225.00
256.45	257.00
225.00	225.00
225.00	193.00
193.00	225.00
257.00	257.00
289.00	321.00
353.00	353.00
383.88	385.00
385.00	384.27
353.00	353.00

5. OPTIMIZACION DE TRAYECTORIA CON SOFTWARE MASTERCAM

En términos prácticos, para llevar a cabo la comparación de la trayectoria obtenida, la empresa INDUSTRIA DE CAUCHOS RECORD S.A.S nos permitió acceder a sus instalaciones en donde un operador calificado para utilizar el software MASTERCAM nos mostró cómo se lleva a cabo este proceso en la industria, para iniciar a elaborar cualquier cavidad es necesario tener la pieza a mecanizar, la cual estaba prediseñada,(ver Figura 65) , posteriormente se procede a efectuar los pasos subsiguientes con el fin de efectuar la generación de la trayectoria y la longitud.

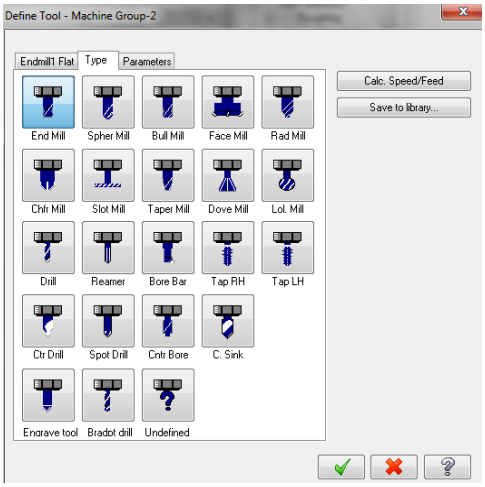
Figura 65. Figura prediseñada



5.1 TRAYECTORIAS GENERADAS POR MASTERCAM.

Con la forma de la figura ya actualizada en el software se procede a seleccionar la herramienta utilizada en el mecanizado y los parámetros de corte (Ver Figura 66).

Figura 66. Selección de la herramienta



A continuación se elige el tipo de cavidad, distancia de retracción de la herramienta, profundidad y dirección de maquinado (ver figura 67 y 68).

Figura 67. Retracción, profundidad y avance de maquinado

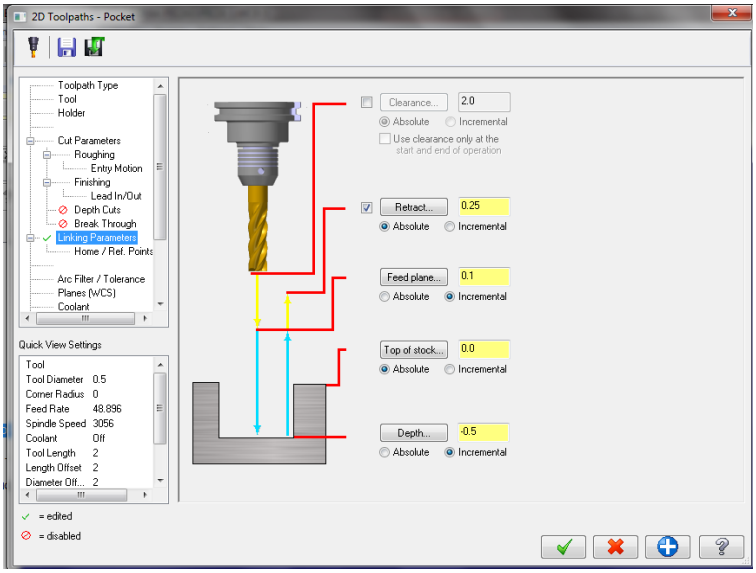
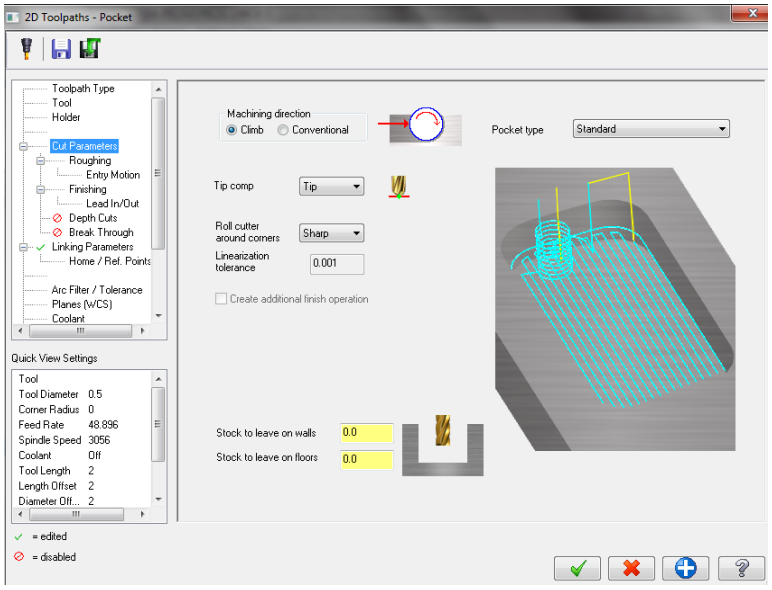
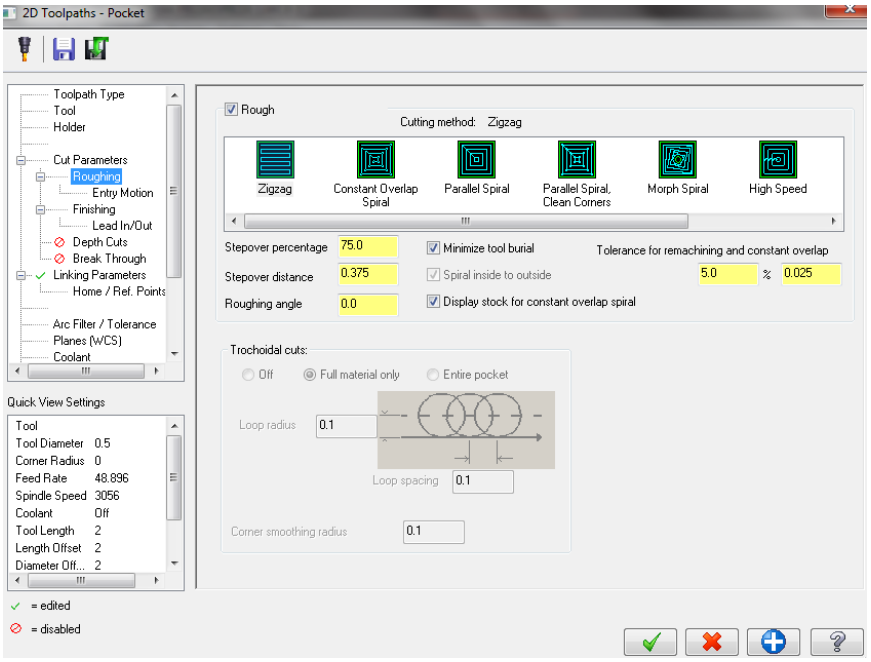


Figura 68. Dirección de maquinado de la herramienta



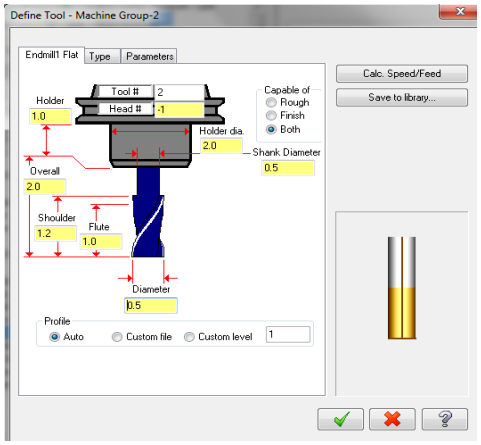
MASTERCAM utiliza 8 trayectorias predeterminadas para realizar el maquinado que son: zig-zag, constant overlap spiral, parallel spiral, parallel spiral clean corner, morph spiral, high speed, one way y trae spiral. Se selecciona en la ventana la trayectoria con la cual se desea trabajar (ver figura 69).

Figura 69.Selección de trayectoria



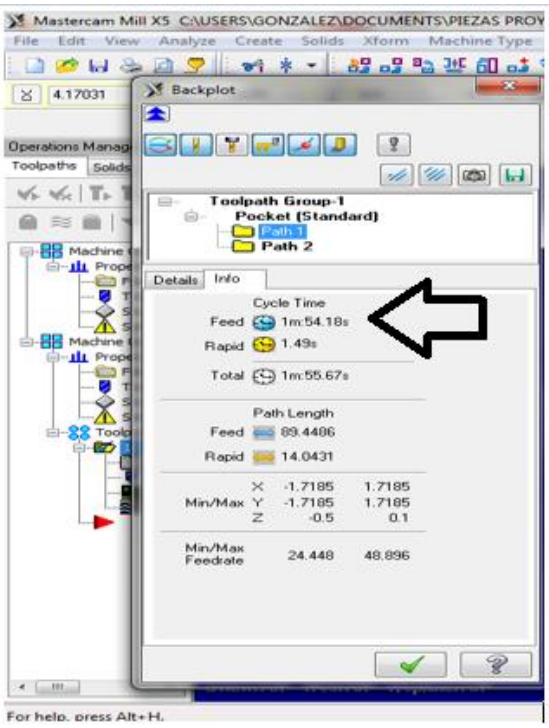
En la siguiente figura se muestran las dimensiones de la herramienta seleccionada en la librería de Mastercam.

Figura 70. Dimensiones de la herramienta



La descripción del procedimiento utilizado por Mastercam para obtener la longitud recorrida por la herramienta para la cavidad inicia seleccionando la herramienta, parámetros de corte y tipo de trayectoria.

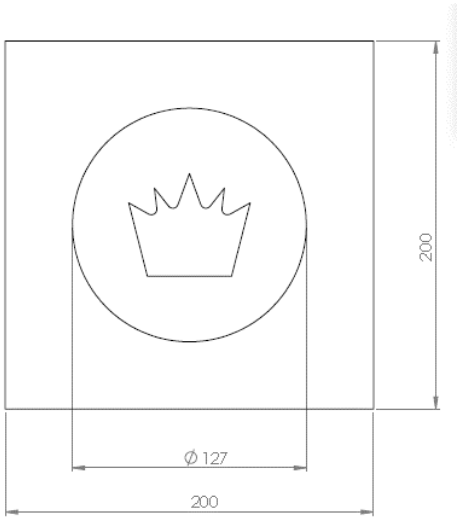
Figura 71.Tiempo de simulación estimado por Mastercam



6. COMPARACION DE METODOS

Los resultados presentados a continuación corresponden a la implementación del programa realizado (NEURAL NET OPTRASOFT 2.0) para las 4 cavidades escogidas ejecutando cada uno de los métodos de solución antes descritos.
Pieza 1 (corona) ver figura 72

Figura 72. Plano de la corona (cavidad 1) con medidas externas (mm)

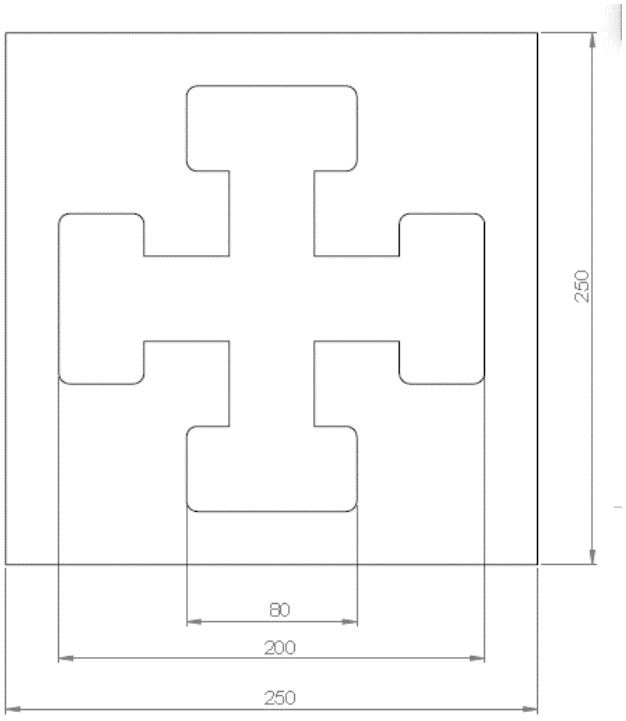


Los resultados de las simulaciones se presentan en la siguiente tabla:

Tabla 5. Resultados corona

Pieza	Diametro (pulgadas)	Interseccion %	Kohonen			Programacion lineal Entera			Hopfield		
			Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución
1	1	30	26,1724	98	8,125"	27,0485	1	1,498	27,7606	33	3,349"
		45	29,3683	144	8,252"	29,4829	1	0,143"	30,2758	72	11,352"
		60	39,0200	225	14,83"	36,8817	1	0,171"	no	no	no
		75	56,2111	417	20,71	54,0303	5	2,26"	no	no	no
	(3/4)	30	35,4217	179	11,999"	34,9216	1	0,088"	no	no	no
		45	59,8137	251	12,638"	537.188	1	0,111"	no	no	no
		60	74,3881	319	15,486"	69,7598	8	1'14,65"	no	no	no
		75	125,8050	1550	1'32,05"	med	med	med	no	no	no
	(1/2)	30	85,9256	429	21,46"	83,3933	21	13'29,927"	no	no	no
		45	113,0000	767	24,783"	med	med	med	no	no	no
		60	162,1920	1376	33,739"	no	no	no	no	no	no
		75	254,4870	5403	1'53,36"	no	no	no	no	no	no
	(1/4)	30	237,0000	2088	56,706"	med	med	med	no	no	no
		45	294,3410	2416	1'37,151"	no	no	no	no	no	no
		60	469,3100	10036	13'23,399"	no	no	no	no	no	no
		75	med	med	med	no	no	no	no	no	no

Figura 73. Plano de la cruz (cavidad 2) con medidas externas (mm)

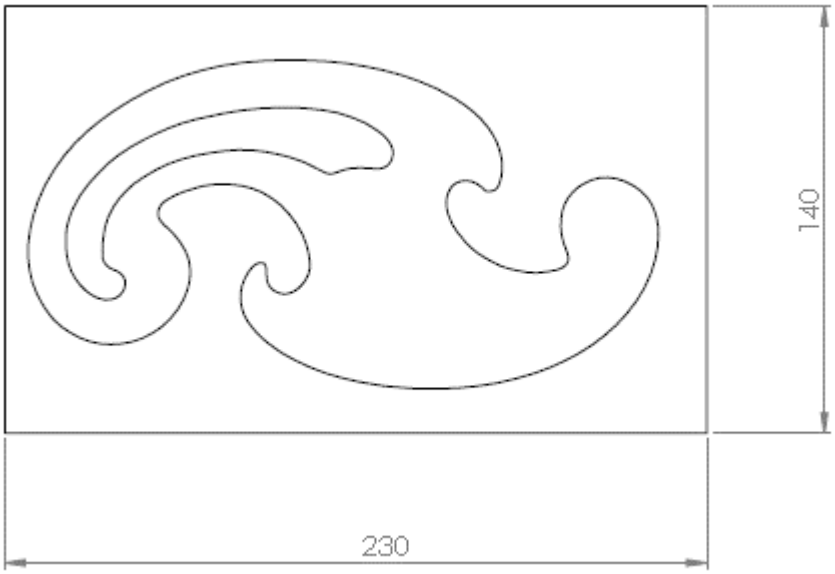


Los resultados de las simulaciones se presentan en la siguiente tabla:

Tabla 6. Resultados cruz

Pieza	Diametro (pulgadas)	Interseccion %	Kohonen			Programacion lineal Entera			Hopfield		
			Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución
2	1'(1/2)	30	41,3309	140	9,195'''	50,2848	5	0,252'''	43,4338	63	6,209''
		45	47,3018	200	14,009''	52,9541	5	0,231'''	45,4274	67	7,814''
		60	66,2770	167	11,182''	69,5982	4	0,376'''	no	no	no
		75	93,6473	403	19,165''	88,906	13	8,497'''	no	no	no
	1'(1/4)	30	66,9580	180	12,188''	71,9116	5	2,071'''	no	no	no
		45	62,0440	205	13,806''	65	5	0,381'''	no	no	no
		60	92,5503	320	14,258''	91,1858	8	0,905'''	no	no	no
		75	124,8710	581	18,786''	118,406	20	21,621'''	no	no	no
	1	30	103,5210	277	11,784''	99,5477	8	0,623'''	no	no	no
		45	105,4140	308	15,071''	101,3084	7	1,847'''	no	no	no
		60	125,9460	507	17,815''	119,772	13	8,37'''	no	no	no
		75	238,0200	7723	1'35,369'''	med	med	med	no	no	no
	(3/4)	30	123,5900	369	17,191''	118,775	11	2,766'''	no	no	no
		45	177,4810	942	27,066''	165,906	18	6,778'''	no	no	no
		60	242,0000	1381	35,12'''	186,922	49	10'40,625	no	no	no
		75	368,0820	3981	1'40,995'''	med	med	med	no	no	no
	(1/2)	30	254,6100	1249	33,301''	med	med	med	no	no	no
		45	320,0370	1905	38,913''	med	med	med	no	no	no
		60	399,6560	3745	1'20,56''	no	no	no	no	no	no
		75	701,6850	8659	11'33,275'''	no	no	no	no	no	no

Figura 74. Plano del curvígrafo (cavidad 3) con medidas externas (mm)

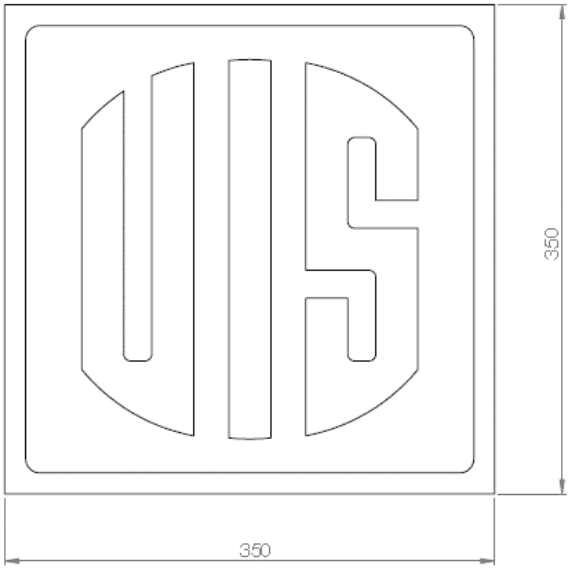


Los resultados de las simulaciones se presentan en la siguiente tabla:

Tabla 7. Resultados curvígrafo

Pieza	Diametro (pulgadas)	Interseccion %	Kohonen			Programacion lineal Entera			Hopfield		
			Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución
3	1'(1/4)	30	17,2995	84	4,629"	18,0584	1	0,115"	20,2703	8	1,585"
		45	16,4250	118	6,841"	21,0887	2	0,2"	15,7625	30	3,146"
		60	20,2121	130	7,546"	24,2329	2	0,141"	21,3087	43	4,332"
		75	29,6105	177	11,068"	32,9568	3	0,237"	no	no	no
	1'	30	20,7174	120	6,132"	25,4622	4	0,189"	20,7173	32	3,239"
		45	27,9609	151	7,985"	30,3529	1	0,072"	30,362	51	5,794"
		60	38,6081	168	8,979"	38,0832	1	0,096"	no	no	no
		75	57,7601	373	14,759"	56,2751	8	1,396"	no	no	no
	(3/4)	30	49,5923	215	11,714"	47,6048	2	0,131"	no	no	no
		45	52,9692	258	14,189"	50,2651	5	0,336"	no	no	no
		60	75,4648	411	14,984"	76,3518	16	4,132"	no	no	no
		75	118,1000	959	22,205"	med	med	med	no	no	no
	(1/2)	30	96,3461	574	20,969"	97,276	11	30,943"	no	no	no
		45	121,5630	826	22,747"	med	med	med	no	no	no
		60	172,1380	1232	26,311"	med	med	med	no	no	no
		75	302,0740	5780	2'13,387"	no	no	no	no	no	no
	(1/4)	30	297,0140	2680	1'17,686"	med	med	med	no	no	no
		45	372,6710	7465	3'54,564"	no	no	no	no	no	no
		60	530,4250	10444	14'11,565"	no	no	no	no	no	no
		75	med	med	med	no	no	no	no	no	no

Figura 75. Plano del Logo UIS (cavidad 4) con medidas externas (mm)



Los resultados de las simulaciones se presentan en la siguiente tabla:

Tabla 8. Resultados Logo UIS

Pieza	Diametro (pulgadas)	Interseccion %	Kohonen			Programacion lineal Entera			Hopfield		
			Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución	Longitud (cm)	Iteraciones	Tiempo de solución
4	1'(1/4)	30	109,612	267	17,232"	113,375	6	1,864"	no	no	no
		45	123,127	434	18,969"	134,201	5	3,268"	no	no	no
		60	152,705	526	18,534"	155,039	16	19,936"	no	no	no
		75	226,050	1132	28,365"	med	med	med	no	no	no
	1	30	212,012	656	19,582"	med	med	med	no	no	no
		45	251,002	894	24,546"	med	med	med	no	no	no
		60	299,595	1961	43,843"	no	no	no	no	no	no
		75	445,632	13771	2'46,702"	no	no	no	no	no	no
	(3/4)	30	291,489	735	22,248"	med	med	med	no	no	no
		45	420,575	1607	39,996"	med	med	med	no	no	no
		60	545,362	4232	1'36,347"	med	mm	med	no	no	no
		75	957,235	12174	14'11,548"	no	no	no	no	no	no
	(1/2)	30	604,218	3378	1'33,525"	med	med	med	no	no	no
		45	909,889	7728	4'17,293"	med	med	med	no	no	no
		60	1087,800	39447	17'4,789"	no	no	no	no	no	no
		75	med	med	med	no	no	no	no	no	no
	(1/4)	30	1645,000	28227	49'26,615"	med	med	med	no	no	no
		45	med	med	med	no	no	no	no	no	no
		60	med	med	med	no	no	no	no	no	no
		75	med	med	med	no	no	no	no	no	no

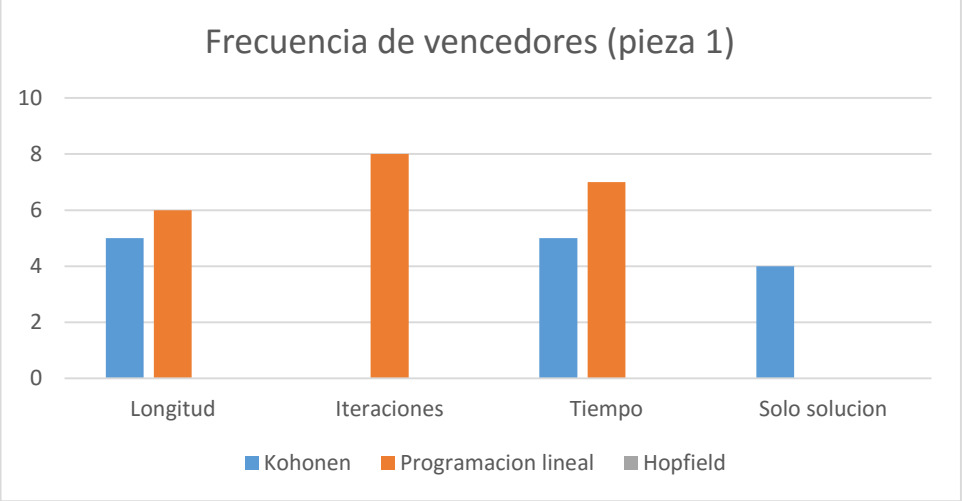
- Análisis de la cavidad 1

En esta cavidad se realizaron pruebas con 4 diámetros diferentes, y en cada diámetro para 4 porcentajes de intersección diferentes (ver tabla 9) para un total de 16 pruebas y los resultados son los siguientes:

Tabla 9. Vencedores cavidad 1

Método	Longitud	Iteraciones	Tiempo	Solo solución
Kohonen	5	0	5	4
Programación lineal	6	8	7	0
Hopfield	0	0	0	0

Figura 76. Diagrama de barras de vencedores (pieza 1)



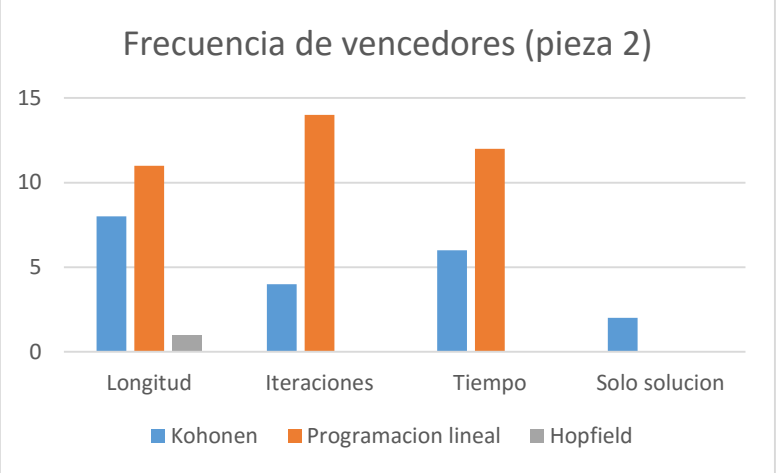
- Análisis de la cavidad 2

En esta cavidad se realizaron pruebas con 5 diámetros diferentes, y en cada diámetro para 4 porcentajes de intersección diferentes (ver tabla 10) para un total de 20 pruebas y los resultados son los siguientes:

Tabla 10. Vencedores cavidad 2

Método	Longitud	Iteraciones	Tiempo	Solo solución
Kohonen	8	4	6	2
Programación lineal	11	14	12	0
Hopfield	1	0	0	0

Figura 77. Diagrama de barras de vencedores (pieza 2)



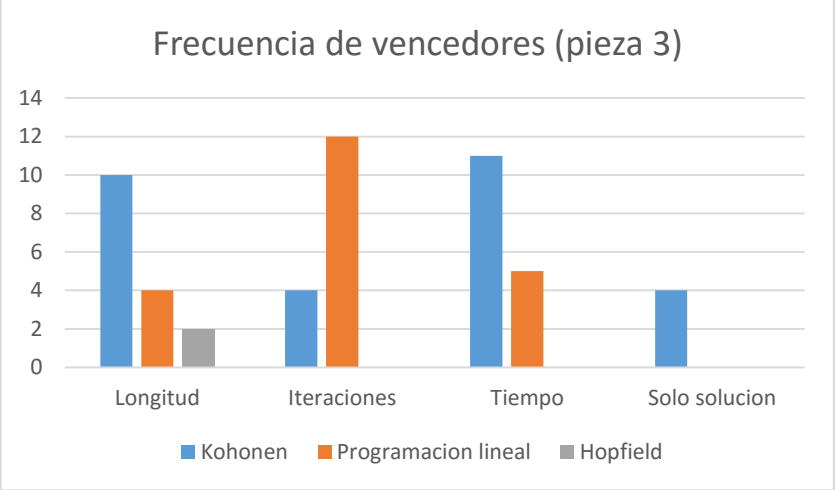
- Análisis de la cavidad 3

En esta cavidad se realizaron pruebas con 5 diámetros diferentes, y en cada diámetro para 4 porcentajes de intersección diferentes (ver tabla 11) para un total de 20 pruebas y los resultados son los siguientes:

Tabla 11. Vencedores cavidad 3

Método	Longitud	Iteraciones	Tiempo	Solo solución
Kohonen	10	4	11	4
Programación lineal	4	12	5	0
Hopfield	2	0	0	0

Figura 78. Diagrama de barras de vencedores (pieza 3)



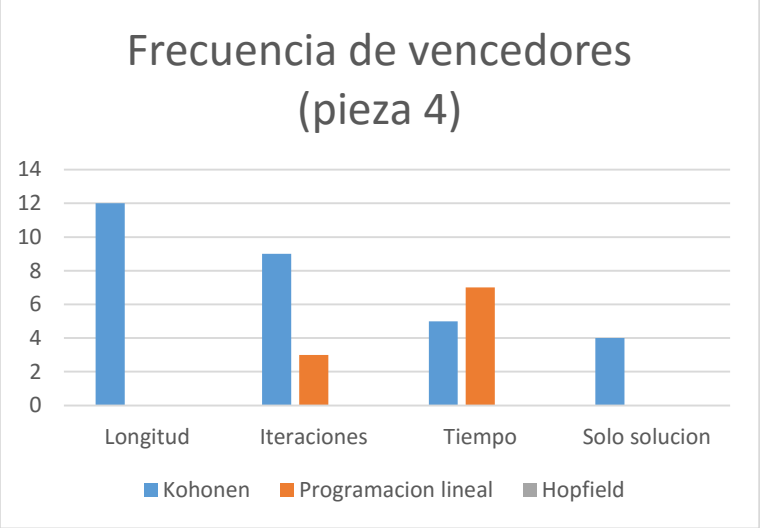
- Análisis de la cavidad 4

En esta cavidad se realizaron pruebas con 5 diámetros diferentes, y en cada diámetro para 4 porcentajes de intersección diferentes (ver tabla #) para un total de 20 pruebas y los resultados son los siguientes:

Tabla 12. Vencedores cavidad 4

Método	Longitud	Iteraciones	Tiempo	Solo solución
Kohonen	12	9	5	4
Programación lineal	0	3	7	0
Hopfield	0	0	0	0

Figura 79. Diagrama de barras de vencedores (pieza 4)



Al hacer un contraste de los resultados obtenido con las frecuencias de vencedores, se puede evidenciar que en diámetros grandes y con un porcentaje de intersección bajo el método de programación lineal entera se presenta como el más eficaz en tiempo e iteraciones, con algunos casos también con longitud de trayectoria.

En los procedimientos de las cuatro piezas analizadas se observa que el método de Hopfield es el segundo mejor a diámetros grandes e intersecciones bajas, por motivo de que genera pocos puntos en la pieza a maquinar, y concordando con la teoría, a medida que el número de puntos aumenta la capacidad de optimización se eleva exponencialmente al punto de no encontrar una solución.

El método de Kohonen se perfila como el más óptimo para solucionar piezas con elevado número de puntos, es decir, diámetros bajos y/o intersecciones altas, y se evidencia en los resultados al ser el único método en las 4 cavidades que al disminuir el diámetro y aumentar la intersección, continua solucionando el problema.

- Análisis con Mastercam

A continuación se presentan los resultados obtenidos al llevar las piezas analizadas a la empresa Industria de Cauchos RECORD S.A.S la cual maneja el software

especializado para el fresado (Mastercam ultima versión), en la cual con la colaboración de un operario se hizo la simulación como si fuera a realizar la pieza obteniendo lo siguiente:

Tabla 13. Resultados Mastercam 2018

Pieza	Longitud desbaste (mm)	Longitud acabado (mm)	Longitud total (cm)	Tiempo total (mecanizado)
1	12865,439	718,484	1358,3923	55' 54,31''
2	31756,962	1323,357	3308,0319	47' 38,83''
3	22820,37	1517,338	2433,7708	33' 22,52''
4	180355,023	16243,569	19659,8592	3h 24' 46,14''

Resulta conveniente aclarar que la realización completa del mecanizado se genera con dos trayectorias, una de desbaste con una fresa grande y una trayectoria de acabado para detalles finales.

Figura 80. Error de trayectoria Mastercam 2018

Al realizar la comparación de las longitudes obtenidas con el Neural Net Optrasoft 2.0 con la longitud de desbaste, se evidencia que con la mayoría de los diámetros la trayectoria más corta es la del Neural Net Optrasoft 2.0.

Durante esta parte práctica se evidencio que algunos métodos de generación de la trayectoria utilizados para hacer una cavidad con algún tipo de islas por MasterCam, no tienen en cuenta estas islas, y entran a mecanizarlas (ver figura 80) por tanto no todos los métodos de generación son viables ya que recortan y establece rutas que no tienen relación con la operación que se desea.

7. CONCLUSIONES

Se cumplió el objetivo del proyecto “COMPARACIÓN DE MÉTODOS DE REDES NEURONALES Y PROGRAMACION LINEAL ENTERA, PARA LA OPTIMIZACIÓN DE LA TRAYECTORIA DE MECANIZADO DE CAVIDADES COMPLEJAS" que fue el desarrollo de un software que mediante la digitalización de un número finito de puntos dentro del área de la cavidad, permite identificar la trayectoria optima de la herramienta en superficies de dos dimensiones, implementado dos modelos fundamentados en redes Neuronales y uno en programación lineal para la solución de trayectorias más cortas para el fresado de diferentes tipos de cavidades, teniendo en cuenta el diámetro de la herramienta, el porcentaje de sobreposición y restricciones propias de cada cavidad.

Al terminar el proyecto se han alcanzado los objetivos planteados al iniciar su desarrollo, elaborando una herramienta informática que facilita la planeación y optimización de la trayectoria de la herramienta en el fresado de cavidades complejas, contribuyendo al desarrollo industrial en las empresas del sector metalmecánico y por medio de ella estimular el desarrollo del conocimiento y uso de técnicas modernas de computación, para procesos metalmecánicos dentro de la industria general.

Al observarse el comportamiento del método de programación lineal se puede concluir que este método es muy eficaz para un número de puntos limitado con diferentes distancias entre ellos, esto gracias a que usa el método de buscar el punto más cercano, en este caso la mayoría de los puntos tienen la misma distancia, por tal motivo, al tener un número elevado de puntos concentrados en una región, la función que realiza la solución del problema encuentra bastantes sub-soluciones factibles (subtures), lo que genera un tiempo prolongado en cada iteración y

mostrando al final muchas trayectorias cortas o en otras palabras, soluciones pequeñas para un problema grande.

Tomando la Red Neuronal de Hopfield, se pudo evidenciar que debido a la función energía que el intenta disminuir, al tener un elevado número de puntos (ciudades), su tiempo de resolución de cada iteración se ve incrementado significativamente y por lo tanto, el tiempo de solución total del problema del agente viajero, durante las pruebas se encontró que este método realiza una solución optima (o encuentra una solución al problema), aproximadamente con un numero de 25 puntos (ciudades).

En todas las cavidades se obtiene que manteniendo el diámetro fijo y aumentar la intersección de fresado, la trayectoria tiende a aumentar aproximadamente en igual porcentaje, debido a que al ubicar los puntos más unidos, la trayectoria puede llegar a mas lugares de la cavidad, de igual manera se puede evidenciar esta tendencia al mantener fija la intersección de fresado y disminuir el diámetro de la fresa, la trayectoria de fresado aumenta según el cambio del tamaño de la fresa (si el diámetro de la fresa se reduce a la mitad, la trayectoria puede aumentar de un 40% a 60%).

Desde el punto de vista financiero el proyecto ofrece resultados que sirven de apoyo en la implementación de un nuevo modelo de trayectoria que tiene gran influencia en los tiempos de maquinado y la vida útil de las herramientas, ofreciendo y garantizando ahorros de tiempo, costos de la herramienta y mayor productividad, al obtener trayectorias más cortas haciendo buen uso del espacio, obedeciendo a principios de eficiencia

Finalmente al realizar la comparación con la empresa *Industria de Cauchos RECORD S.A.S* la cual maneja un software especializado para el fresado (Mastercam 2018 Última Generación), se encontró que los métodos implementados en el presente proyecto a través del software desarrollado, superan la mayor parte

de los métodos que realiza el Mastercam en todas las piezas, encontrando la trayectoria más corta y respetando las islas de las cavidades, además la optimización del tiempo aplicada en Mastercam depende de la experiencia y habilidad del operario para mejorarlo, que a diferencia de la propuesta del presente proyecto se realiza aplicando algoritmos de optimización desarrollados en el software Neural Net Optrasoft 2.0.

BIBLIOGRAFIA

BLACK, C. Stewart, CHILES, Vic, LISSAMAN, A. J., MARTIN, S. J., Principios de ingeniería de manufactura. México: CECSA. 1999.

CABRERA RODRÍGUEZ, Sonia I. Aplicación de la programación Lineal a la agronomía. [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://matematicas.uclm.es/ita-cr/web_matematicas/trabajos/248/Programacion_lineal.pdf

GALVIZ PARRA, Edgar y GONZALEZ ALMEIDA, Fabio Leonel. Aplicación de redes neuronales para la optimización de la trayectoria de la herramienta en el fresado de cavidades complejas. Bucaramanga, 2012, Trabajo de grado (Ingeniero Mecánico). Universidad Industrial de Santander. Facultad de ingenierías Físico mecánicas. Disponible en el catálogo en línea de la Biblioteca de la Universidad Industrial de Santander: <http:// http://tangara.uis.edu.co/>

GROOVER, Mikell P. Fundamentos de manufactura moderna, materiales, procesos y sistemas. México: Prentice Hall. 1997. ISBN-13:978-970-10-6240-1

KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.724. ISBN 978-970-26-1026-7

HILERA, José R. y MARTÍNEZ, Víctor J. Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones. Madrid: Alfaomega. 2000.

HOPFIELD, J. J. y TANK, D. W. Neural computation of decisions in optimization problem. Biological cybernetics Vol. 52 (1985); p 141-152.

KOHONEN, Teuvo. Self organization and associative memory. New York: Springer-Verlag. 1989.

SUK-HWAN, Suh y YANG-SOO, Shin. Neural network modeling for tool path planning of the rough cut in complex pocket milling. Journal of Manufacturing Systems. 1996.

UNIVERSIDAD DE BUENOS AIRES, Departamento de matemáticas: Optimización Combinatoria. [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.dm.uba.ar/materias/investigacion_operativa/2011/2/capit_5.pdf

UNIVERSIDAD DE MURCIA, Programación Lineal. [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

UNIVERSITAT DE VALÈNCIA, Programación Lineal Entera. [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en: http://www.uv.es/~sala/Clase14.pdf

ANEXOS

Anexo A. COMPLEMENTO MARCO TEÓRICO FRESADO

OTRAS OPERACIONES DE FRESADO Y CORTADORES DE FRESADO

Se utilizan muchas otras operaciones y cortadores de fresado para maquinar piezas de trabajo.

En el fresado combinado de fresas paralelas, se montan dos o más cortadores en un eje para maquinar dos superficies paralelas en la pieza (figura 81a). El fresado de forma produce perfiles curvados empleando cortadores que tienen dientes muy afilados (figura 81b). Dichos cortadores también sirven para cortar dientes de engranes. Las operaciones de acanalado y cortado se realizan usando *cortadores circulares*, como se muestra en la figura 81c y d, respectivamente. Los dientes se pueden alternar en forma ligera, como los de una hoja de sierra, para proveer espacio al cortador cuando produce ranuras profundas. Las *sierras de cortado* son relativamente delgadas, por lo general menores a 5 mm ($\frac{3}{16}$de pulgada) Los *cortadores de ranura T* se utilizan para fresar ranuras T, como las que se encuentran en las mesas de trabajo de máquinas herramienta para sujetar piezas de trabajo. Como se muestra en la figura 82a, primero se fresa una ranura con una fresa frontal; después el cortador maquina en un pase el perfil completo de la ranura T.

Los *cortadores de ranura para chaveta* se utilizan en la elaboración de ranuras de chavetas semicilíndricas (Woodruff) para ejes. Los *cortadores de fresado angular* (ángulo simple o doble ángulo) se emplean para producir superficies cónicas con varios ángulos.

Las *fresas huecas* (figura 82b) son huecas y se montan en un eje; esto permite utilizar el mismo eje para cortadores de diferentes tamaños. El uso de fresas huecas es similar al de fresas frontales. El fresado con un solo diente de corte montado en un husillo de alta velocidad se conoce como *fresado de un solo filo*; por lo general, sirve en operaciones de fresado de careado y mandrinado. A esta herramienta se le puede dar forma como una herramienta de corte de una sola punta y puede colocarse en varias posiciones radiales en el husillo.

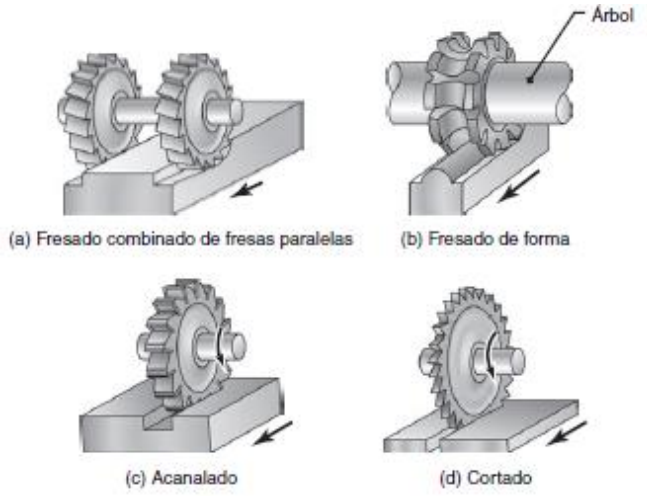
PORTAHERRAMIENTAS

Los cortadores de fresado se clasifican en cortadores de árbol o cortadores tipo zanco. Los cortadores de árbol se montan en un árbol (ver figuras 81 y 85a) para operaciones como fresado periférico, de careado, combinado de fresas paralelas y de forma. En el cortador tipo zanco, el cortador y el zanco se hacen de una pieza; las fresas frontales son los ejemplos más comunes. Las fresas frontales pequeñas poseen zancos rectos, pero las más grandes tienen zancos cónicos para montarlas mejor en el husillo de la máquina, a fin de resistir las fuerzas y torque más altos comprendidos en el corte. Los cortadores con zancos rectos se montan en boquillas de pinzas o en sujetadores especiales de fresas frontales; los que tienen zancos cónicos se montan en portaherramientas cónicos.

Además de los tipos mecánicos, también existen portaherramientas y ejes hidráulicos.

La rigidez de los cortadores y portaherramientas es importante para proporcionar calidad a la superficie y reducir la vibración y el traqueteo durante las operaciones de fresado.

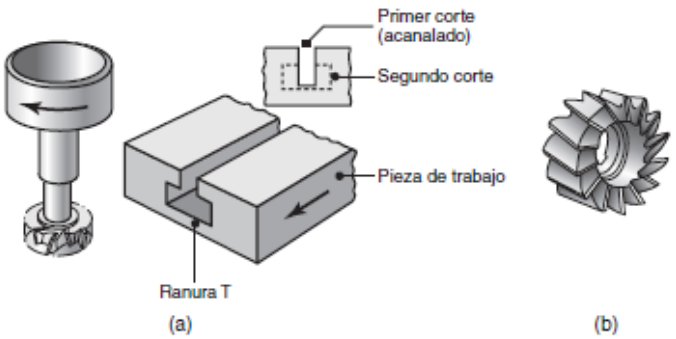
Figura 81. Tipos de cortadores



Cortadores para *(a)* fresado combinado de fresas paralelas; *(b)* fresado de forma; *(c)* acanalado, y *(d)* cortado con un cortador de fresado.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.734. ISBN 978-970-26-1026-7

Figura 82. Tipos de corte



(a) Corte de ranura T con un cortador de fresado. *(b)* Fresa hueca.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.735. ISBN 978-970-26-1026-7.

CAPACIDADES DEL PROCESO DE FRESADO

Además de las diversas características de los procesos de fresado descritas hasta ahora, sus capacidades incluyen parámetros como acabado superficial, tolerancias dimensionales, capacidad de producción y consideraciones de costos. En la tabla 14 se presentan los intervalos convencionales de velocidades y avances de corte para el fresado. Dependiendo del material de la pieza de trabajo y sus condiciones, el material de la herramienta de corte y los parámetros del proceso, las velocidades de corte varían ampliamente, desde 30 hasta 3000 m/min (90 a 10,000 pies/min). Por lo general, el avance por diente va de alrededor de 0.1 mm (0.004 pulgada) a 0.5 mm (0.02 pulgada) y las profundidades de corte van de 1 mm a 8 mm (0.04 a 0.30 pulgada). Para recomendaciones sobre el fluido de corte.

En la tabla 15 se proporciona una guía general de resolución de problemas; en las figuras 83 y 84 se ilustran los últimos cuatro casos de dicha tabla. El contragolpeteo comprende marcas de avance doble producidas por el extremo posterior del cortador. En la tabla 14 se observa que algunas recomendaciones (como el cambio de parámetros de fresado o de herramientas de corte) son más fáciles de seguir que otras (como los cambios de ángulos de las herramientas y la geometría del

cortador, además de la rigidez de los husillos y los dispositivos de sujeción del trabajo).

LINEAMIENTOS DE DISEÑO Y OPERACIÓN PARA FRESADO

Algunos factores adicionales relativos a dichas operaciones son los siguientes:

Hasta donde sea posible, deben utilizarse cortadores de fresado estándares, dependiendo de las características de diseño de las partes. Tienen que evitarse cortadores especiales costosos.

Deben especificarse chaflanes o biseles en vez de radios; si se especifican radios, será difícil hacer que coincidan diversas superficies de intersección.

Deben evitarse las cavidades y bolsas internas con esquinas puntiagudas, dada la dificultad para fresarlas, ya que los dientes o insertos de corte tienen un radio de filo finito. Cuando sea posible, el radio de la esquina debe coincidir con la geometría del cortador de fresado.

Las piezas de trabajo deben ser suficientemente rígidas para minimizar las deflexiones que pudieran producir las fuerzas de sujeción y corte.

Los lineamientos para evitar la vibración y el traqueteo en el fresado son similares a los del torneado. Además, deben considerarse las siguientes prácticas:

Los cortadores deben montarse tan cerca como sea posible de la base del husillo, para reducir las deflexiones de las herramientas.

Los portaherramientas y dispositivos de fijación deben ser tan rígidos como sea posible.

En casos de vibración y traqueteo, deben modificarse las condiciones de forma y proceso de la herramienta y utilizarse cortadores con menos dientes de corte o con espaciado aleatorio de dientes.

Tabla 14. Recomendaciones generales para operaciones de fresado

Material	Herramienta de corte	Condiciones iniciales de propósito general		Variedad de condiciones	
		Avance mm/diente (pulgadas /diente)	Velocidad m/min (pies/min)	Avance mm/diente (pulgadas /diente)	Velocidad m/min (pies/min)
Aceros de bajo carbono y de maquinado libre	Carburo sin recubrimiento, carburo recubierto, cermets	0.13–0.20 (0.005–0.008)	120–180 (400–600)	0.085–0.38 (0.003–0.015)	90–425 (300–1400)
Aceros aleados					
Blandos	Cermets sin recubrimiento, recubiertos	0.10–0.18 (0.004–0.007)	90–170 (300–550)	0.08–0.30 (0.003–0.012)	60–370 (200–1200)
Duros	Cermets, PcBN	0.10–0.15 (0.004–0.006)	180–210 (600–700)	0.08–0.25 (0.003–0.010)	75–460 (250–1500)
Hierro fundido, gris					
Blando	Cermets sin recubrimiento, recubiertos, SiN	0.10–10.20 (0.004–0.008)	120–760 (400–2500)	0.08–0.38 (0.003–0.015)	90–1370 (300–4500)
Duro	Cermets, SiN, PcBN	0.10–0.20 (0.004–0.008)	120–210 (400–700)	0.08–0.38 (0.003–0.015)	90–460 (300–1500)
Acero inoxidable, Austenítico	Cermets sin recubrimiento, recubiertos	0.13–0.18 (0.005–0.007)	120–370 (400–1200)	0.08–0.38 (0.003–0.015)	90–500 (300–1800)
Aleaciones de alta temperatura Base níquel	Cermets sin recubrimiento, recubiertos, SiN, PcBN	0.10–0.18 (0.004–0.007)	30–370 (100–1200)	0.08–0.38 (0.003–0.015)	30–550 (90–1800)
Aleaciones de titanio	Cermets sin recubrimiento, recubiertos	0.13–0.15 (0.005–0.006)	50–60 (175–200)	0.08–0.38 (0.003–0.015)	40–140 (125–450)
Aleaciones de aluminio					
Maquinado libre	PCD sin recubrimiento, recubiertos	0.13–0.23 (0.005–0.009)	610–900 (2000–3000)	0.08–0.46 (0.003–0.018)	300–3000 (1000–10,000)
Alto silicio	PCD	0.13 (0.005)	610 (2000)	0.08–0.38 (0.003–0.015)	370–910 (1200–3000)
Aleaciones de cobre	PCD sin recubrimiento, recubiertos	0.13–0.23 (0.005–0.009)	300–760 (1000–2500)	0.08–0.46 (0.003–0.018)	90–1070 (300–3500)
Plásticos	PCD sin recubrimiento, recubiertos	0.13–0.23 (0.005–0.009)	270–460 (900–1500)	0.08–0.46 (0.003–0.018)	90–1370 (300–4500)

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.736. ISBN 978-970-26-1026-7.

Las profundidades de corte, *d*, van generalmente de 1 a 8 mm (0.04 a 0.3 pulgadas).

PcBN*

PCD**.

* Nitruro de boro cúbico policristalino

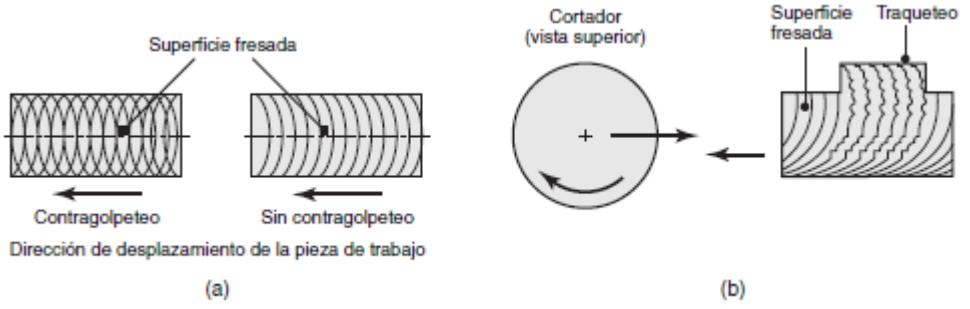
** Diamante policristalino

Tabla 15. Guía general de resolución de problemas para operaciones de fresado

Problema	Causas probables
Ruptura de la herramienta	El material de la herramienta carece de tenacidad, ángulos inapropiados de la herramienta, parámetros de maquinado demasiado elevados.
Desgaste excesivo de la herramienta	Parámetros de maquinado demasiado elevados, material inadecuado de la herramienta, ángulos inapropiados de la herramienta, fluido de corte inadecuado.
Acabado rugoso de la superficie	Avance por diente demasiado alto, muy pocos dientes en el cortador, herramienta astillada o desgastada, acumulación del borde, vibración y traqueteo.
Tolerancias demasiado amplias	Falta de rigidez del husillo y sujeción del trabajo, aumento excesivo de temperatura, herramienta desafilada, virutas congestionando el cortador.
Superficie bruñida de la pieza de trabajo	Herramienta desafilada, profundidad de corte demasiado baja, ángulo de alivio radial muy pequeño.
Contragolpeteo	Herramientas de corte desafiladas, inclinación en husillo del cortador, ángulos negativos de la herramienta.
Marcas de traqueteo	Rigidez insuficiente en el sistema; vibraciones externas; avance, profundidad de corte y anchura de corte demasiado grandes.
Formación de rebabas	Filos de corte desafilados o demasiado honeados, ángulo incorrecto de entrada o salida, avance y profundidad de corte demasiado altos, forma incorrecta del inserto.
Ruptura	Ángulo de avance demasiado bajo, geometría incorrecta del filo de corte, ángulo incorrecto de entrada o salida, avance y profundidad de corte demasiado altos.

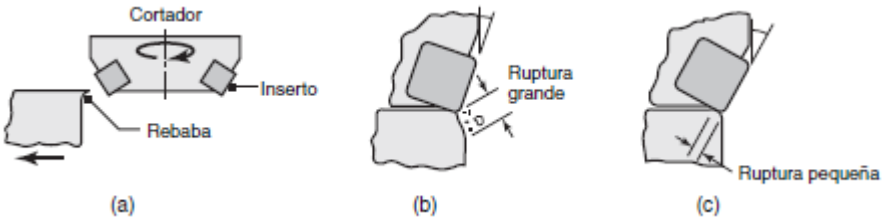
Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.737. ISBN 978-970-26-1026-7.

Figura 83. Características de superficie maquinada en el fresado de careado.



Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.737. ISBN 978-970-26-1026-7.

Figura 84. Defectos de los filos en el fresado de careado



Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.737. ISBN 978-970-26-1026-7.

FRESADORAS

Dado que tienen la capacidad de realizar varias operaciones de corte, las fresadoras se encuentran entre las máquinas herramienta más versátiles y útiles. La primera fue construida en 1820 por Eli Whitney (1765-1825). En la actualidad existe una gran selección de fresadoras con diversos usos. A continuación se describen las características de las fresadoras estándar. Sin embargo, muchas de estas máquinas y operaciones están siendo reemplazadas con controles por computadora y centros de maquinado.

Maquinas tipo columna y codo. Utilizadas para operaciones de fresado de propósito general, las máquinas tipo columna y codo son las fresadoras más comunes. El husillo en el que se monta el cortador de fresado puede ser horizontal (figura 85a), para fresado periférico, o vertical, para operaciones de fresado de careado y frontal, mandrinado y taladrado (figura 85b). Los componentes básicos de estas máquinas son:

Mesa de trabajo: en la que se sujeta la pieza de trabajo utilizando ranuras T. La mesa se mueve longitudinalmente en relación con las fresas paralelas.

Carro: soporta la mesa y puede moverse en dirección transversal.

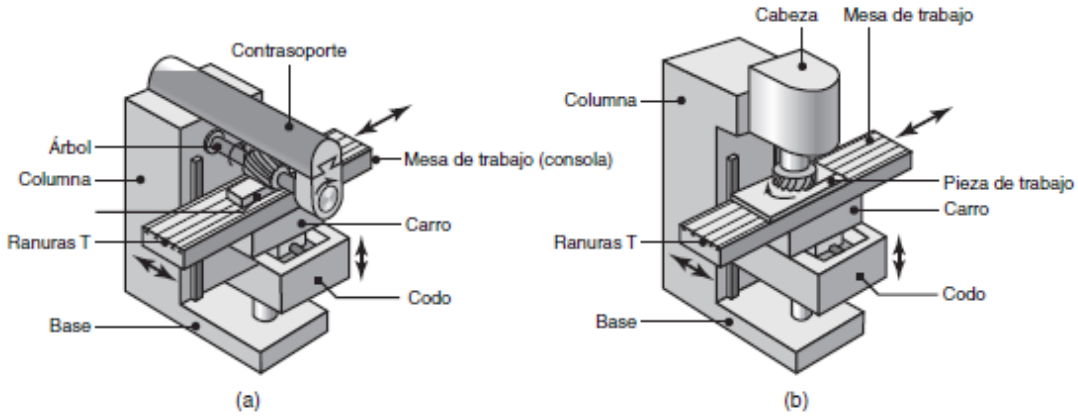
Codo: soporta el carro y da movimiento vertical a la mesa, de manera que la profundidad de corte puede ajustarse y es posible acomodar piezas de trabajo con diversas alturas.

Brazo superior: se utiliza en máquinas horizontales; es ajustable para acomodar diferentes longitudes de eje.

Cabezal: contiene el husillo y el sujetador del cortador. En máquinas verticales, la cabeza puede fijarse o ajustarse verticalmente y girarse en un plano vertical sobre la columna para cortar superficies cónicas.

Las fresadoras simples tienen tres ejes de movimiento, que por lo general se mueven manual o mecánicamente. En las fresadoras tipo columna y codo universales, la mesa puede girar en el plano horizontal. De esta manera se pueden maquinar formas complejas (como canales helicoidales a diversos ángulos) para producir partes como engranes, brocas, machuelos y cortadores.

Figura 85. Fresadoras



Esquema de *(a)* fresadora tipo columna y codo de husillo horizontal, y *(b)* fresadora tipo columna y codo de husillo vertical.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.738. ISBN 978-970-26-1026-7.

Fresadoras tipo bancada. En las máquinas tipo bancada, la mesa de trabajo se monta directamente en la bancada, que reemplaza al codo y puede moverse sólo en forma longitudinal (figura 86). Estas máquinas no son tan versátiles como otros tipos, pero tienen alta rigidez y por lo general se utilizan para trabajo de alta producción. Los husillos pueden ser horizontales o verticales y tipo dúplex o triples (con dos o tres husillos), para maquinado simultáneo de dos o tres superficies de una pieza de trabajo.

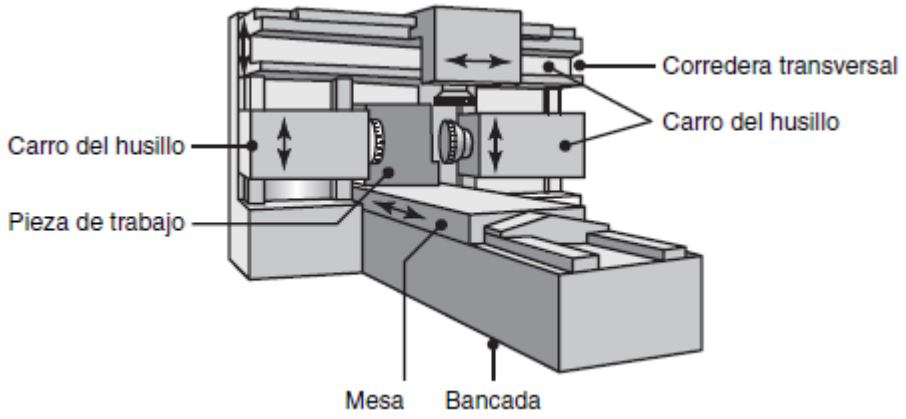
Otros tipos de fresadoras. Existen muchos otros tipos de fresadoras. Las *fresadoras tipo cepilladora*, que son similares a las máquinas tipo bancada, están equipadas con diversas cabezas y cortadores para fresar diferentes superficies. Se utilizan en piezas de trabajo pesadas y son más eficaces que las cepilladoras cuando se usan con propósitos semejantes. Las *máquinas de mesa giratoria* son similares a las fresadoras verticales y están equipadas con una o más cabezas para operaciones de fresado de careado.

Las máquinas de fresado están siendo reemplazadas con rapidez por *máquinas de control numérico por computadora* (CNC), que son versátiles y tienen la capacidad de fresar, taladrar, mandrinar y machuelear con precisión repetitiva (figura 87). También existen *fresadoras de perfiles*, que cuentan con cinco ejes de movimiento (figura 88); obsérvese los tres movimientos lineales y los dos movimientos angulares de los componentes de la máquina.

Dispositivos y accesorios de sujeción de trabajo. La pieza de trabajo a fresar debe sujetarse con seguridad a la mesa de trabajo para resistir las fuerzas de corte y evitar su deslizamiento durante el fresado. Con este propósito se utilizan diversos montajes y tornillos de banco. Se montan y sujetan a la mesa mediante las ranuras T mostradas en la figura 85a y b. Los tornillos de banco se emplean para trabajo de pequeña producción en partes pequeñas. Los montajes se usan para trabajo de mayor producción y se pueden automatizar por medios mecánicos e hidráulicos.

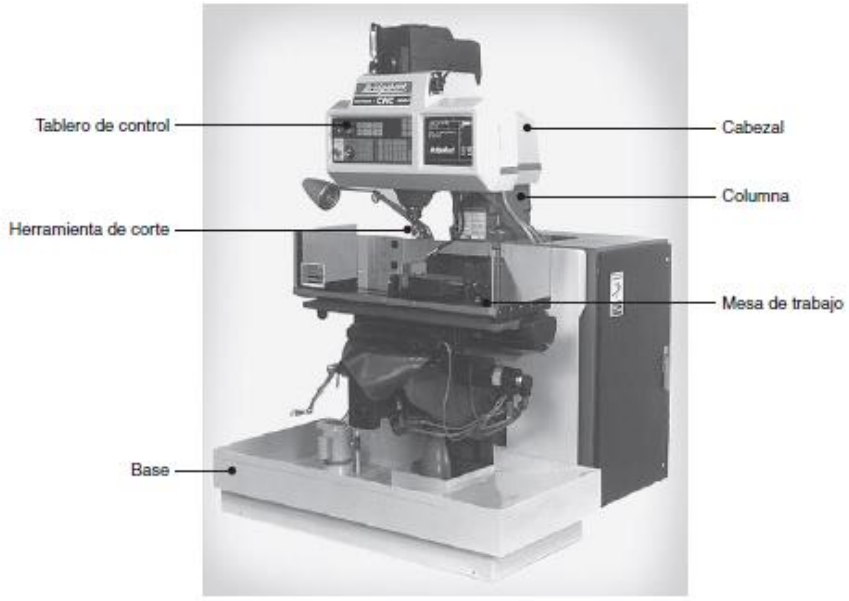
Los accesorios para fresadoras incluyen diversos soportes y accesorios para el cabezal de la máquina (así como para la mesa de trabajo) diseñados con el propósito de adaptarlos a diferentes operaciones de fresado. Por lo general, el accesorio más común en los talleres ha sido el *cabezal divisor (de índice) universal*. De accionamiento manual, este aditamento gira (indexa) la pieza de trabajo a ángulos especificados entre los pasos individuales de maquinado. Por lo común se ha utilizado para fresar partes con superficies poligonales y para maquinar dientes de engranes. En la actualidad, los cabezales divisores sólo se usan para pequeñas cantidades en talleres de trabajo; han sido reemplazados por los controles CNC y los centros de maquinado.

Figura 86. Esquema de una fresadora tipo bancada.



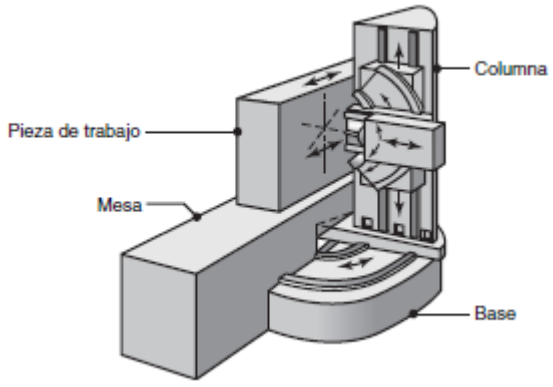
Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.738. ISBN 978-970-26-1026-7.

Figura 87. Fresadora de husillo vertical de control numérico computadora (CNC).



Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.740. ISBN 978-970-26-1026-7.

Figura 88. Esquema de una fresadora de perfiles de cinco ejes.



Obsérvese que existen tres movimientos lineales principales y dos angulares de los componentes de la máquina.

Fuente: KALPAKJIAN, Serope y SHMID, Steven R. Manufactura, ingeniería y tecnología. México: Pearson Educación, 2008. p.740. ISBN 978-970-26-1026-7.

Anexo B. COMPLEMENTO MARCO TEÓRICO DE PROGRAMACIÓN LINEAL

EL MÉTODO SIMPLEX.

El método gráfico que se utilizó para resolver problemas de programación lineal ha mostrado que cuando la región factible de un problema de programación lineal no es vacía, siempre tiene puntos esquina (o vértices) y, lo que es más importante, que si el problema tiene solución óptima finita siempre existe, al menos, una en un punto vértice de la región factible. Este resultado es clave en el desarrollo del método simplex que resuelve los problemas de programación lineal. Como los problemas con más de tres variables de decisión no se pueden visualizar (vivimos en un espacio tridimensional) no se tiene más remedio que caracterizar algebraicamente a los puntos vértice de la región factible. Esto se consigue convirtiendo todas las restricciones de desigualdad del problema (excepto las de no negatividad) en ecuaciones y dando una propiedad algebraica que sólo la cumplen los vértices de la región factible.

El método simplex resuelve el problema en iteraciones. Parte de un vértice de la región factible y en cada iteración se traslada a un nuevo vértice con mejor valor potencial de la función objetivo. El proceso acaba cuando llega a un vértice cuyo valor objetivo no puede ser mejorado.

El método gráfico identifica los puntos factibles de vértice de la región factible. Estos son candidatos a solución óptima y hay un número finito de ellos. Por su parte, el método algebraico determina las soluciones básicas factibles del sistema de ecuaciones y, como se sabe, el número de soluciones básicas de un sistema de ecuaciones es finito, de manera que el número de candidatos (soluciones básicas factibles) a ser solución óptima es finito. El método gráfico utiliza las curvas de nivel de la función objetivo para determinar el vértice óptimo. En el método algebraico se utiliza la función objetivo para seleccionar a la solución básica factible óptima, entre todas las candidatas.

Las reglas para seleccionar la variable de entrada y de salida se llaman condiciones de optimalidad y de factibilidad, respectivamente. A continuación se resumen dichas condiciones y los pasos del método simplex.

Condición de optimalidad: La variable de entrada en un problema de maximización (minimización) es la variable no básica que tenga el coeficiente más negativo (positivo) en el renglón cero (renglón z). Si hay empate se rompe arbitrariamente. Se llega al óptimo en la iteración en la que todos los coeficientes de las variables no básicas en el renglón z son mayores o iguales a cero (menores o iguales a cero)

Condición de factibilidad: Tanto en los problemas de maximización como en los de minimización la variable de salida es la variable básica asociada a la mínima razón entre los lados derecho y los coeficientes positivos de la columna pivote, sin contar la entrada de tal columna en el renglón z. Si hay empate se rompe arbitrariamente.

Los pasos del método simplex son:

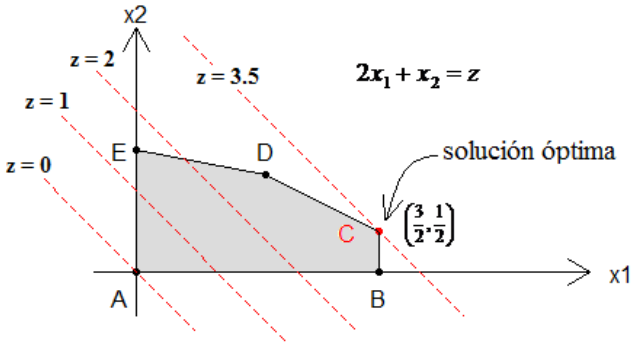
Paso 1. Determinar una solución básica factible inicial.
Paso 2. Seleccionar una variable de entrada aplicando la condición de optimalidad. Si no hay variable de entrada detenerse. La última tabla proporciona una solución óptima.
Paso 3. Seleccionar una variable de salida aplicando la condición de factibilidad. Seleccionar el elemento pivote.
Paso 4. Determinar una nueva solución básica factible con la operación de pivoteo. Ir al paso 2.
Recapitulando lo hecho.

Se parte de un problema de programación lineal,

$$(P)\begin{cases} \text{Maximizar} \quad z = 2x_1 + x_2 \\ \text{sujeto a} \\ \qquad 3x_1 + 8x_2 \le 12 \\ \qquad x_1 + x_2 \le 2 \\ \qquad 2x_1 \le 3 \\ \qquad x_1, x_2 \ge 0 \end{cases}$$

Problema que se puede representar y resolver gráficamente,

Figura 89. Representación Solución optima



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://ocw.um.es/ingenierias/complementos-de-algebra-lineal/practicas-1/programacion-lineal-jfa.pdf

Solución óptima única:

Se sabe que en un problema de maximizar (minimizar) se llega al óptimo en la iteración en la que todos los coeficientes de las variables no básicas en el renglón *z* son mayores o iguales a cero (menores o iguales a cero). Si todos los coeficientes de las variables no básicas en el renglón *z* son mayores estrictamente a cero (menores estrictamente a cero) la solución óptima a la que se ha llegado es única.

Caracterización de soluciones. Solución: Un conjunto de valores x es una solución del problema si cumple el sistema de ecuaciones Ax = b

Solución factible

Un conjunto de valores x es una solución factible del problema si cumple el sistema de ecuaciones Ax = b (es solución) y cumple que x ≥ 0, es decir todos los valores son no negativos.

Solución linealmente independiente:

Una solución x es linealmente independiente si las columnas de la matriz A correspondientes a las variables con valor no nulo son linealmente independientes.

Solución linealmente dependiente

Una solución x es linealmente dependiente si las columnas de la matriz A correspondientes a las variables con valor no nulo son linealmente dependientes.

Características de las soluciones linealmente independientes

Dado que en un problema de programación lineal n > m, si el rango de la matriz A es m:

El número máximo de componentes no nulas de una solución linealmente independiente es, m y

El número mínimo de componentes no nulas de una solución linealmente dependiente típicamente es m+ 1

TIPOS DE PROGRAMACIÓN LINEAL

Programación lineal entera. Son aquellos en que todas las variables pueden tomar valores enteros. También se distinguen dentro de estos los problemas totalmente enteros como aquellos en que las variables y todos los coeficientes del problema son enteros.

La programación lineal entera[4] es una técnica matemática utilizada en la investigación de operaciones, que permite la optimización de una función objetivo a través de la aplicación de diversas restricciones a sus variables
Un problema de programación lineal entera es un problema de programación lineal con la restricción adicional de que algunas de las variables deben tomar valores enteros.

Una gran variedad de problemas combinatorios pueden ser planteados como problemas de programación lineal entera.

MÉTODO DE RAMIFICACIÓN Y ACOTACIÓN (Branch and Bound)

El método de ramificación y acotación, más conocido por su nombre en inglés Branch and Bound[5] , recibe su nombre precisamente por las dos técnicas en las que basa su desarrollo, que son la ramificación y la acotación.
El método de ramificación y acotación comienza por resolver el PLA, de modo que si la solución al PLA verifica las condiciones de integridad, entonces también es la solución al problema entero, en caso contrario se comienza con la ramificación del problema.

La Ramificación. La ramificación consiste en dividir cada problema en dos nuevos subproblemas, obtenidos mediante la imposición de restricciones excluyentes que dividen el conjunto de oportunidades del problema original en dos partes, pero eliminando en ambas partes la solución no entera del problema original.

---

[4] [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.dm.uba.ar/materias/investigacion_operativa/2011/2/capit_5.pdf

[5] [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en: http://www.uv.es/~sala/Clase14.pdf

Cuando en la solución al PLA una variable que ha de ser entera xi toma el valor xbi no entero, entonces se generan a partir de dicho valor dos restricciones xi ≤ [xbi] y xi ≥ [xbi]+1 (siendo [xbi] la parte entera por defecto de xbi), que añadidas cada uno por separado al problema original, da lugar a dos nuevos subproblemas. Se explicara este proceso a través de un ejemplo particular:

Se considera el siguiente problema

$$Max\ F(x)\ =\ 4x1\ +\ 5x2\ (\mathbf{1})$$
$$s.a.\ 2x1\ +\ x2\ \leq\ 8$$
$$x2\ \leq\ 5$$
$$x1, x2\ \geq\ 0\ y\ enteras$$

La solución al PLA, prescindiendo de la condición de que las variables han de ser enteras es

$$x1\ =\ 1{,}5, x2\ =5\ y\ F(x)\ =\ 31$$

Como dicha solución no verifica las condiciones de integridad se elige la variable x1 que no es entera y a partir de ella se generan dos restricciones

$$x1\ \leq\ 1 \qquad y \qquad x1\ \geq\ 2$$

Que añadidas cada una de ellas al problema original dan lugar a dos nuevos subproblemas que serían los siguientes:

$$Max\ F(x)\ =\ 4x1\ +\ 5x2\ (1.1) \qquad Max\ F(x)\ =\ 4x1\ +\ 5x2\ (1.2)$$
$$s.a.\ 2x1\ +\ x2\ \leq\ 8 \qquad\qquad s.a.\ 2x1\ +\ x2\ \leq\ 8$$
$$x2\ \leq\ 5 \qquad\qquad x2\ \leq\ 5$$
$$x1\ \leq\ 1 \qquad\qquad x1\ \geq\ 2$$
$$x1, x2\ \geq\ 0 \qquad\qquad x1, x2\ \geq\ 0$$

De este modo se han eliminado todas las posibles soluciones no enteras del conjunto de oportunidades tales que 1< x1 < 2.

El proceso se repite con cada uno de los dos subproblemas obtenidos, los cuales dará lugar a otros dos subproblemas cada uno de ellos y así sucesivamente hasta que en todos los subproblemas tengan solución entera o infectable.

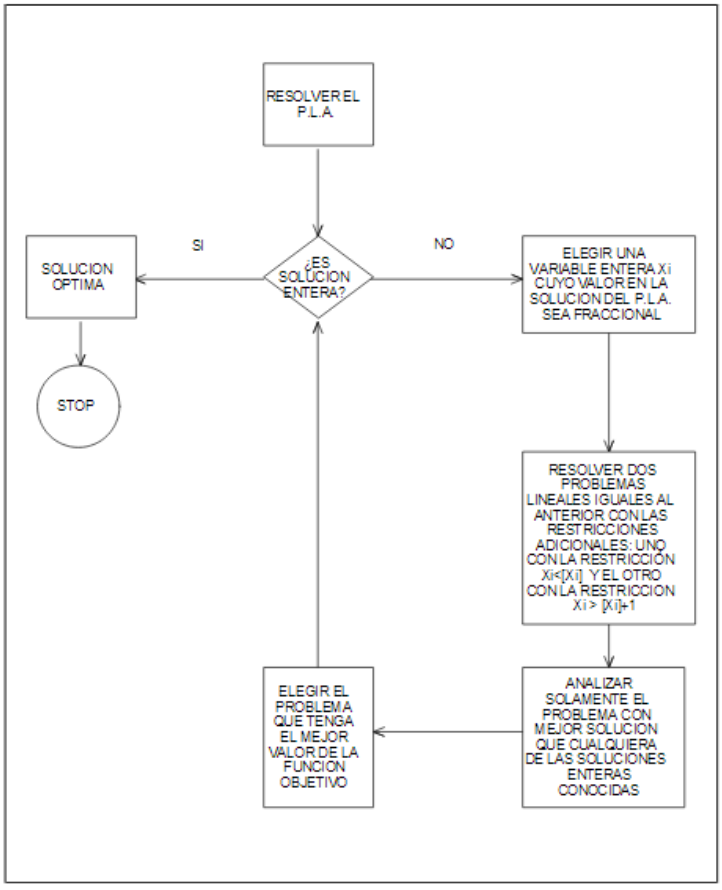
Utilizando únicamente la ramificación, el número de subproblemas a resolver crece exponencialmente, por este motivo para evitar el tener que resolver todos los subproblemas, la ramificación se combina con la acotación.

La Acotación. La acotación se basa en el hecho de que dado que los conjuntos de oportunidades del subproblema 1.1 (S11) y del subproblema 1.2 (S12) son a su vez subconjuntos del conjunto de oportunidades del problema 1 (S1) la solución óptima de los dos subproblemas siempre será inferior (problema de máximo o superior para problemas de mínimo) que la solución óptima del problema 1 por ser los conjuntos de elección menores.

Así pues, el proceso de acotación consiste, para problemas de máximo, en tomar como cota inferior aquella solución entera con mayor valor de la función objetivo obtenida y dado que cualquier otro subproblema con solución no entera se sabe que al ramificarlo dará como resultado valores de la función objetivo menores o iguales, permite descartar como subproblemas a ramificar todos aquellos que tengan como solución óptima un valor de la función inferior a la cota establecida.

De este modo se reduce el número de subproblemas a ramificar y por lo tanto el tiempo necesario para la resolución de los problemas enteros.
El proceso a seguir en la resolución de problemas enteros mediante el método de ramificación y acotación se resume en la siguiente figura o esquema algorítmico:

Figura 90. Esquema del algoritmo de ramificación y acotación

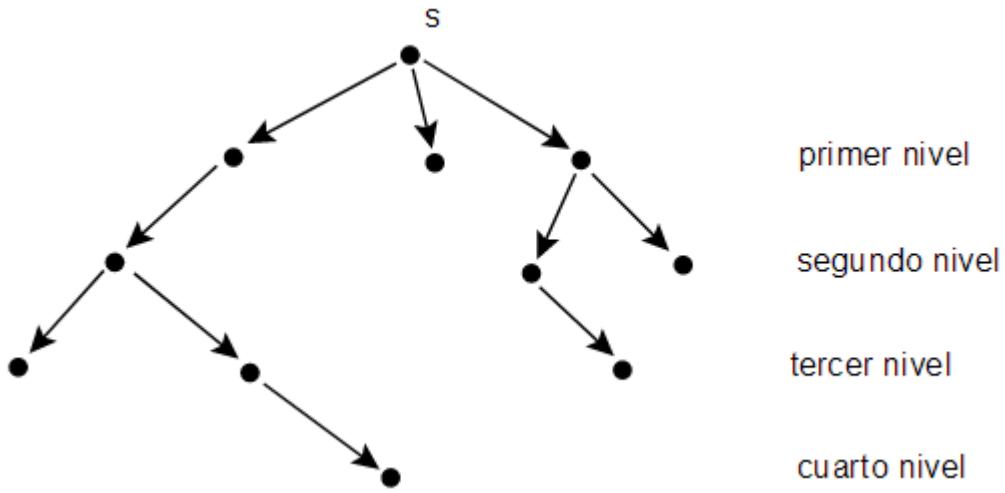


Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.uv.es/~sala/Clase14.pdf

A continuación se presentan 2 ejemplos de este método:

Ejemplo 1.

Figura 91. Árbol dirigido con raíz y cuatro niveles



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.dm.uba.ar/materias/investigacion_operativa/2011/2/capit_5.pdf

Si un árbol dirigido con raíz es binario y tiene n niveles entonces tiene $2^{n+1} - 1$ vértices.

Se considerara el siguiente problema: dado un árbol dirigido con raíz s, donde cada vértice x tiene asignado un costo c(x) que satisface $c(x) \leq c(y)$ para toda rama (x, y) se quiere hallar una hoja de mínimo costo.

Se describirá un algoritmo, conocido como Branch and Bound, que resuelve este problema. El procedimiento utiliza una mezcla de backtracking (volver al vértice anterior para examinar alguno de sus hijos que todavía no ha sido examinado) y un criterio particular de poda que consiste en eliminar toda la descendencia de un vértice x cuando se satisfaga que $c(x) \geq c(h)$ para alguna hoja h encontrada previamente (es decir, cuando ninguna hoja descendiente de x puede ser la solución óptima del problema).

0. $L = \{s\}$, $c = \infty$.

1. Si x es, de los elementos de L, el ultimo que ingresó, $L = L - \{x\}$.

Calcular c (x). Si $c(x) \geq c$, vaya a 4.

2. Si x no es una hoja, L = L ∪ {Hijos de x}, vaya a 4.

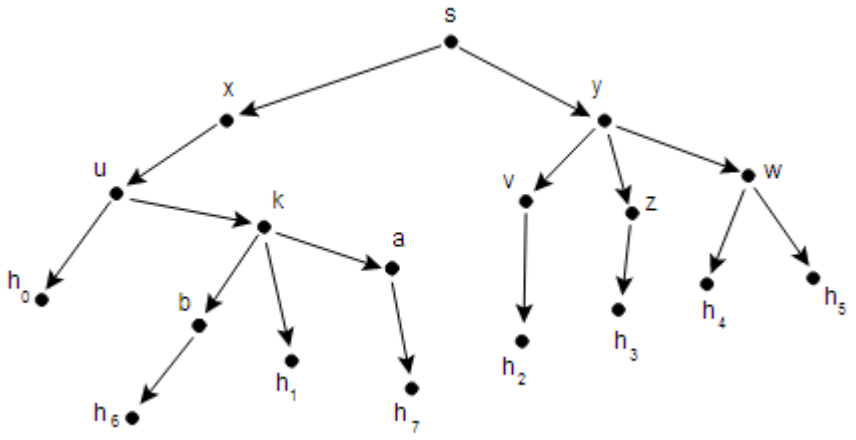
3. 3. h = x, c = c (x).

4. Si L f = ∅ vaya a 1.

5. STOP.

Ejemplo. Se Considera el árbol dirigido con raíz s

Figura 92. Árbol con raíz s



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.dm.uba.ar/materias/investigacion_operativa/2011/2/capit_5.pdf

Donde $c(s) = 3$, $c(x) = 4$, $c(y) = 4$, $c(z) = 4$, $c(u) = 7$, $c(v) = 8$, $c(w) = 7$, $c(k) = 10$, $c(a) = 12$, $c(b) = 13$,$c(h_0) = 8$, $c(h_1) = 11$, $c(h_2) = 9$, $c(h_3) = 6$, $c(h_4) = 8$ y $c(h_5) = 11$, $c(h_6) = 14$, $c(h_7) = 14$.

Aplicando el algoritmo se obtiene:

0. $L = \{s\}$, $c = \infty$.

1. $L = \emptyset$, $c(s) = 3$.

2. $L = \{x, y\}$, go to 4.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x\}$, $c(y) = 4 < \infty = c$.

2. $L = \{x, v, z, w\}$, go to 4.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x, v, z\}$, $c(w) = 7 < \infty = c$.

2. $L = \{x, v, z, h_4, h_5\}$, go to 4.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x, v, z, h_4\}$, $c(h_5) = 11 < \infty = c$.

3. $h = h_5$, $c = 11$.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x, v, z\}$, $c(h_4) = 8 < 11 = c$.

3. $h = h_4$, $c = 8$.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x, v\}$, $c(z) = 4 < 8 = c$.

2. $L = \{x, v, h_3\}$, go to 4.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x, v\}$, $c(h_3) = 6 < 8 = c$.

3. $h = h_3$, $c = 6$.

4. $L \neq \emptyset$, go to 1.

1. $L = \{x\}$, $c(v) = 8 \geq 6 = c$, go to 4.

4. $L \neq \emptyset$, go to 1.

1. $L = \emptyset$, $c(x) = 4 < 6 = c$.

2. $L = \{u\}$, go to 4.

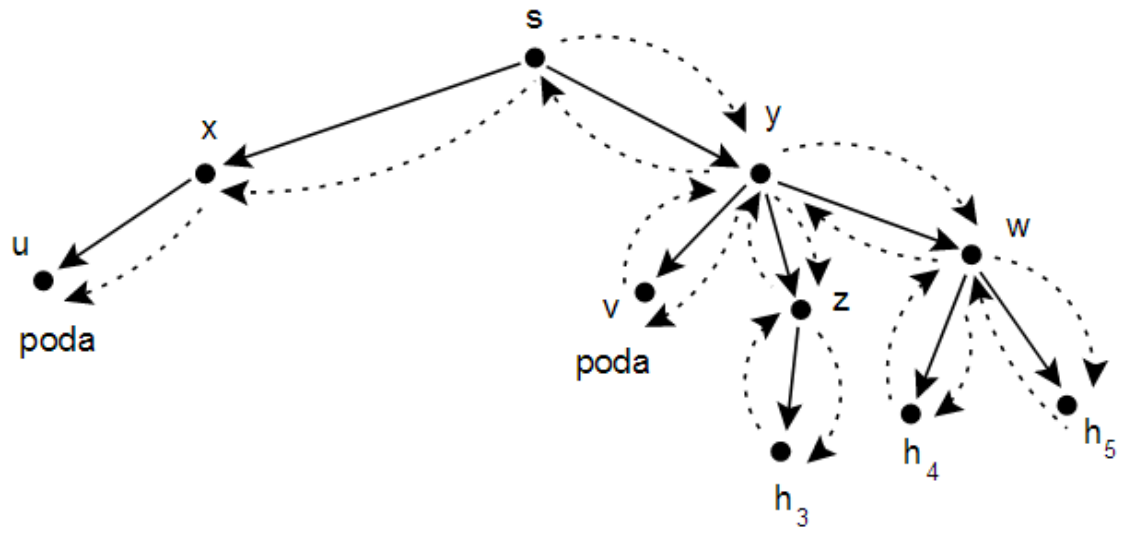
$L \neq \emptyset$, go to 1.

1. $L = \emptyset$, $c(u) = 7 \geq 6 = c$, go to 4. 4. $L \neq \emptyset$.

STOP.

El siguiente gráfico es una interpretación de los pasos seguidos por el algoritmo en este ejemplo.

Figura 93. Interpretación de los pasos del algoritmo



Fuente: [En línea]. (Recuperado en 30 septiembre 2017.) Disponible en http://www.dm.uba.ar/materias/investigacion_operativa/2011/2/capit_5.pdf

Los vértice son examinados en el orden en que indican las flechas punteadas descendentes. Cada flecha punteada ascendente indica un backtracking. El valor de *c* en cada iteración del algoritmo es el costo de la hoja más barata encontrada hasta el momento y *h* guarda la información de cuál es la hoja cuyo costo es *c* (cuando el algoritmo encuentra una hoja de costo menor que el valor de *c* presente en ese momento, actualiza *c* y *h*).

Si al examinar un nodo *i* resulta que $c(i) \geq c$ entonces toda su descendencia es podada ya que ninguna hoja que sea un descendiente de *i* puede tener costo menor que *c*, es decir, costo menor que el de la hoja más barata hallada hasta ese momento.

Como una hoja no es examinada por el algoritmo solo cuando es seguro que no puede ser una solución óptima entonces en alguna iteración del algoritmo una hoja de mínimo costo es examinada. Cuando el algoritmo encuentra la primer hoja de mínimo costo los valores de *h* y *c* son actualizados ya que su costo es menor que el valor de *c* que estaba presente en ese momento, y a partir de allí *c* y *h* permanecen constantes hasta terminar el algoritmo ya que ninguna otra hoja encontrada más

tarde puede tener costo menor que el de esta hoja de mínimo costo. Luego, al terminar el algoritmo la hoja que se encuentra almacenada en h es una hoja de mínimo costo y $c = c(h)$.

Luego, en este caso, la hoja de mínimo costo es $h = h_3$, con costo $c = 6$.

En este ejemplo la función $c(x)$ estaba dada explícitamente. Esto no es así en general, sino que es una función que se debería calcular.

En el caso en que la función c satisfaga $c(x) \geq c(y)$ para toda rama (x, y), podemos resolver el problema de hallar una hoja de máximo costo utilizando el algoritmo que resulta de reemplazar, en el paso 0., $c = \infty$ por $c = -\infty$ y, en el paso 1., la condición $c(x) \geq c$ por la condición $c(x) \leq c$, es decir, el algoritmo

0. $L = \{s\}$, $c = -\infty$.

1. Sea x es el último vértice que ingreso en L, $L = L - \{x\}$. Calcular $c(x)$. Si $c(x) \leq c$, go to 4.

2. Si x no es una hoja, $L = L \cup \{\text{hijos de } x\}$, go to 4.

3. $h = x$, $c = c(x)$.

4. Si $L \neq \emptyset$ go to 1

5. STOP

Sin embargo, a pesar de disponer de herramientas y de lenguajes de programación diseñados expresamente para el desarrollo de "máquinas inteligentes", existe un problema de fondo que limita enormemente los resultados que se pueden obtener: estas "máquinas" se implementan sobre ordenadores basados en la filosofía de funcionamiento expuesta inicialmente por Von Neumann, y se apoyan en una descripción secuencial del proceso de tratamiento de la información. El elevado nivel y desarrollo de éstos ordenadores por espectacular y complejo que haya llegado a ser, no deja de seguir la línea antes expuesta: una máquina puramente mecánica que es capaz de realizar tareas mecánicas (de cálculo, ordenación o control) de forma increíblemente rápida, pero incapaz de obtener resultados aceptables cuando se trata de tareas sencillas, por ejemplo, para un ser humano de corta edad (reconocimiento de formas, habla, etc.).

La otra línea de investigación ha tratado, desde los orígenes de la humanidad, de aplicar los principios físicos que rigen en la naturaleza para obtener máquinas que realizaran los trabajos pesados en nuestro lugar. Así, por ejemplo, los motores de vapor o explosión, emplean un determinado tipo de combustión para obtener la energía que necesitan, al igual que lo hacen los seres vivos. De igual manera se puede pensar respecto a la forma y capacidad de razonamiento del ser humano; se puede intentar obtener máquinas con esta capacidad basadas en el mismo principio de funcionamiento (o en algo que tenga una cierta similitud con dicho principio).

No se trata de construir máquinas que compitan con los seres humanos, sino que realicen ciertas tareas de "rango intelectual" con que ayudarle, lo que supone un principio básico de la inteligencia artificial. Por otro lado, los sistemas que se lleguen a desarrollar no van a suponer la desaparición de los ordenadores, tal como hoy los entendemos, por lo menos en aquellas tareas en las que están mejor dotados incluso que los seres humanos.

Las primeras explicaciones teóricas sobre el cerebro y el pensamiento ya fueron dadas por algunos antiguos filósofos griegos como Platón (427-347 a.C.) y Aristóteles (384-422 a.C.).Las mismas ideas sobre el proceso mental también las mantuvieron Descartes (1596-1650) y los filósofos empiristas del siglo XVIII.

La clase de las llamadas máquinas cibernéticas, a la cual la computación neuronal pertenece, tiene más historia de lo que generalmente se cree: Heron el alejandrino construyó un autómata hidráulico sobre el 100 a.C. Además, se han construido numerosos modelos de animales para demostrar el comportamiento necesidad-adaptación sobre diferentes condiciones de vida como, por ejemplo, las numerosas versiones del ratón en el laberinto.

Alan Turing, en 1936, fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación; sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas. Ellos modelaron una red neuronal simple mediante circuitos eléctricos. Otro importante libro en los inicios de la teoría de redes neuronales fue el escrito en 1949 por Donald Hebb: La organización del comportamiento, en el que se establece una conexión entre psicología y fisiología.

En 1957, Frank Rosenblatt comenzó el desarrollo del Perceptrón, un modelo de red neuronal del cual hablaremos más adelante. El perceptrón es la más antigua red neuronal y se usa hoy en día de varias formas para la aplicación como reconocedor de patrones; este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones era capaz de reconocer otros similares aunque no se le hubieran presentado anteriormente. Sin embargo, tenía una serie de limitaciones, quizá la más conocida era su incapacidad para resolver el problema

de la función OR-exclusiva y, en general, no era capaz de clasificar clases no separables linealmente.

En 1959, Bernard Widrow y Marcial Hoff, de Stanford, desarrollaron el modelo ADALINE (ADAptative LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) y se ha usado comercialmente durante varias décadas.

Uno de los mayores investigadores de las redes neuronales desde los años 60 hasta nuestros días es Stephen Grossberg (Universidad de Boston). A partir de su extenso conocimiento fisiológico, ha escrito numeros libros y desarrollado modelos de redes neuronales. Estudió los mecanismos de la percepción y la memoria. Grossberg realizó en 1967 una red, Avalancha, que consistía en elementos discretos con actividad que varía con el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades tales como reconocimiento continuo del habla y aprendizaje del movimiento de los brazos de un robot.

En 1969 surgieron numerosas críticas que frenaron, hasta 1982, el crecimiento que estaban experimentando las investigaciones sobre redes neuronales. Marvin Minsky y Seymour Papert, del Instituto Tecnológico de Massachusetts (MIT) publicaron un libro: Perceptrons [Minsky 69], que, además de contener un análisis matemático detallado del perceptrón, consideraban que la extensión a perceptrones multinivel (el perceptrón original solo poseía una capa) era completamente estéril. Las limitaciones del perceptrón eran importantes, sobre todo, su incapacidad para resolver muchos problemas interesantes. Muchas de las investigaciones dieron un giro hacia la Inteligencia Artificial, que prometía más por aquel entonces.

A pesar del libro Perceptrons, algunos investigadores continuaron su trabajo. Tal fue el caso de James Anderson que desarrolló un modelo lineal llamado Asociador Lineal que consistía en unos elementos integradores lineales (neuronas) que

sumaban sus entradas. Este modelo se basa en el principio de que las conexiones entre neuronas son reforzadas cada vez que están activadas. Anderson diseñó una potente extensión del Asociador Lineal llamado "Brain-State-in-a-Box" (BSB) .

En Europa y Japón las investigaciones también continuaron. Kunihiko Fukushima desarrolló el Neocognitrón, un modelo de red neuronal para el reconocimiento de patrones visuales. Teuvo Kohonen, un ingeniero eléctronico de la universidad de Helsinki, desarrolló un modelo similar al de Anderson pero independientemente.

En 1982, numerosos eventos coincidieron que hicieron resurgir el interés por las redes neuronales. John Hopfield presentó su trabajo sobre redes neuronales en la Academia Nacional de las Ciencias. En el trabajo describe con claridad y rigor matemático una red a la que ha dado su nombre, que es una variación del Asociador Lineal, pero además mostró como tales redes pueden trabajar y qué pueden hacer. Además, en 1982 se celebró la U.S.-Japan Joint Conference on Cooperative/Competitive Neural Networks y Fujitsu comenzó el desarrollo en "computadores pensantes" para aplicaciones en robótica.

En 1985, el Instituto Americano de Física comenzó lo que ha sido la reunión anual Neural Networks for Computing. Esta ha sido la primera de muchas otras. En 1987, el IEEE celebró la primera conferencia internacional sobre redes neuronales con más de 1.800 asistentes y 19 nuevos productos mostrados. En el mismo año se formó la Internatonal Neural Network Society (INNS) bajo la iniciativa y dirección de Grossberg en U.S.A., Kohonen en Finlandia y Amari en Japón; en menos de dos años, la INNS tenía más de 3.000 socios. A partir de este momento, el interés por este área de la I+D se ha ido incrementando de forma notable, como lo demuestran tanto el número de congresos y reuniones científicas especializadas, como la aparición de revistas científicas de calidad contrastada dentro del área, así como el interés demostrado por diversos tipos de empresas en utilizar esta tecnología para desarrollar aplicaciones concretas.

En 1988, del espíritu de cooperación en esta nueva tecnología resultó la unión del IEEE y de la INNS; la International Joint Conference on Neural Networks (IJCNN) produjo, en 1989, 430 artículos, 63 de los cuales enfocados a una aplicación. La IJCNN de Enero de 1990, en Washington, incluyó una hora de concierto de música realizada por redes neuronales. La alternativa europea es la International Conference on Artificial Neural Networks (ICANN), que comenzó su andadura en Septiembre de 1991 y, actualmente, está organizada por la Sociedad Europea de Redes Neuronales. También merece una referencia aparte la reunión anual Neural Information Processing Systems (NIPS) celebrada en Denver (Colorado) desde 1987 y que probablemente represente el nivel más alto de calidad desde el punto de vista científico.

Definición de red neuronal

Es necesario destacar que tales "computadores neuronales" no ejecutan las típicas instrucciones máquina de los computadores digitales, a menos que estén hechos para emular el comportamiento de las redes neuronales físicas. En principio, la operación de proceso básico realizada por todos los procesadores elementales es una operación análoga de transformación de sus señales de entrada.

En las redes neuronales biológicas, las células neuronales (neuronas) corresponden a los elementos de proceso anteriores. Las interconexiones se realizan por medio de las ramas de salida (axones) que producen un número variable de conexiones (sinapsis) con otras neuronas (o quizá con otras partes como músculos y glándulas). Las redes neuronales son sistemas de simples elementos de proceso fuertemente interconectados.

La compleja operación de las redes neuronales es el resultado de abundantes lazos de realimentación junto con "no linealidades" de los elementos de proceso y

cambios adaptativos de sus parámetros, que pueden definir incluso fenómenos dinámicos muy complicados.

Una peculiaridad de las redes neuronales biológicas es su tamaño: en todo el sistema nervioso central hay del orden de 10^{11} neuronas, pero el número de interconexiones es aún mayor, probablemente sobre las 10^{15}. No parece posible programar las funciones de dicho sistema de acuerdo con un plan principal, además teniendo en cuenta que el tamaño y la estructura de la red está cambiando radicalmente durante y después de la niñez, cuando está ya en uso.

Es verdad que ciertos caracteres textuales de la red son inherentes y durante la ontogénesis las proyecciones neuronales crecen aproximadamente hacia aquellos lugares en los cuales serán necesitados más tarde. En otras palabras, la distribución de los recursos y los caminos de comunicación más importantes, están formados de acuerdo con un plan genético, mientras que el resto del "programming" en especial la memoria, debe ser adquirida después de nacer.

Programar tal red puede significar sólo dos cosas: a) Las estructuras de interconexión entre las células son alteradas, y b) Las "fuerzas" de estas interconexiones son cambiadas. Parece que existen bastantes estrategias claras de cómo cambiar las fuerzas en la dirección correcta, mientras que cambios en las interconexiones son más difíciles de definir porque suelen tener efectos radicales en el comportamiento de la red, especialmente en lo concerniente a la operación secuencial y las funciones jerárquicas.

Es muy difícil imaginar cómo una red tan enorme puede ser programada. Una posibilidad, con relación a los subsistemas sensoriales podría ser que la estructura del sistema, o el proceso dinámico definido por él, de alguna forma tiende a imaginar las experiencias sensoriales de otros acontecimientos.

Otra función importante del sistema nervioso es definir acciones que son parte del comportamiento, y controlar el estado del organismo en relación con su entorno.

Mientras que las representaciones internas en las cuales está basado el comportamiento pueden ser derivadas de las entradas de forma bastante directa, las definiciones de las salidas deben ser basadas en estrategias completamente diferentes. De hecho, apenas existe otra posibilidad para programar las acciones que aplicar el principio de "retroceso y empuje" para alterar los mecanismos que son responsables de ellas. Algún tipo de "backpropagation" de la información es, por tanto, necesario.

Por otro lado, el significado y la calidad de las acciones deben ser juzgados no desde los movimientos inmediatos, sino desde la realización de criterios que tienen en cuenta el resultado requerido, a veces bastante indirectamente. A menudo, las acciones son solamente corregidas si se realizan con cierta frecuencia, donde el mecanismo que lo provoque debe contener circuitos que definan tales frecuencias y que se cambien en relación con los resultados aprendidos. Está claro que programar las acciones es un proceso mucho más indirecto que programar las representaciones internas; el factor aleatorio no puede ser evitado.

Programando las funciones de entrada y de salida nos lleva sólo a una operación de comportamiento en la que el estímulo y la respuesta son consideradas muy relevantes. Ciertamente es posible realizar autómatas bastante complejos y comportamientos necesidad-conducta de este modo. Sin embargo, se mantiene la expectativa sobre la posibilidad de que las redes neuronales puedan actuar como ordenadores para algunos problemas abstractos y también donde la computación se realice en el estado interno de la red.

Ventajas de las redes neuronales

Aprendizaje adaptativo. La capacidad de aprendizaje adaptativo es una de las características más atractivas de las redes neuronales. Esto es, aprenden a llevar a cabo ciertas tareas mediante un entrenamiento con ejemplos ilustrativos. Como las redes neuronales pueden aprender a diferenciar patrones mediante ejemplos y entrenamiento, no es necesario que se elaboren modelos a priori ni se necesitan especificar funciones de distribución de probabilidad.

Las redes neuronales son sistemas dinámicos autoadaptativos. Son adaptables debido a la capacidad de autoajustarse de los elementos procesales (neuronas) que componen el sistema. Son dinámicos pues son capaces de estar constantemente cambiando para adaptarse a las nuevas condiciones.

En el proceso de aprendizaje, los enlaces ponderados de las neuronas, se ajustan de manera que se obtengan unos resultados específicos. Una red neuronal no necesita un algoritmo para resolver un problema, ya que ella puede generar su propia distribución de los pesos de los enlaces mediante el aprendizaje. También existen redes que continúan aprendiendo a lo largo de su vida después de completado el período inicial de entrenamiento

La función del diseñador es únicamente la obtención de la arquitectura apropiada. No es problema del diseñador el cómo la red aprenderá a discriminar; sin embargo, si es necesario que desarrolle un buen algoritmo de aprendizaje que proporcionará la capacidad de discriminar de la red mediante un entrenamiento con patrones.

Autoorganización. Las redes neuronales usan su capacidad de aprendizaje adaptativo para autoorganizarse la información que reciben durante el aprendizaje y/o la operación. Mientras que el aprendizaje es la modificación de cada elemento procesal, la autoorganización consiste en la modificación de la red neuronal completa para llevar a cabo un objetivo específico.

Cuando las redes neuronales se usan para reconocer ciertas clases de patrones, ellas se autoorganizan la información usada. Por ejemplo, la red llamada back-propagation se creará su propia representación característica mediante la cual puede reconocer ciertos patrones.

Esta autoorganización provoca la generalización: facultad de las redes neuronales de responder apropiadamente cuando se les presenta datos o situaciones a los que no habían sido expuestas anteriormente. El sistema puede generalizar la entrada para obtener una respuesta. Esta característica es muy importante cuando se tienen que solucionar problemas en los cuales la información de entrada es poco clara; además, permite que el sistema de una solución incluso cuando la información de entrada está especificada de forma incompleta.

Tolerancia a fallos. Las redes neuronales son los primeros métodos computacionales con la capacidad inherente de tolerancia a fallos. Comparados con los sistemas computacionales tradicionales, los cuáles pierden su funcionalidad en cuanto sufren un pequeño error de memoria, en las redes neuronales, si se produce un fallo en un pequeño número de neuronas, aunque el comportamiento del sistema se ve influenciado, sin embargo, no sufre una caída repentina.

Hay dos aspectos distintos respecto a la tolerancia a fallos: primero, las redes pueden aprender a reconocer patrones con ruido, distorsionados o incompletos, esta es una tolerancia a fallos respecto a los datos. Segundo, ellas pueden seguir realizando su función (con cierta degradación) aunque se destruya parte de la red.

La razón por la que las redes neuronales son tolerantes a los fallos es el que tienen su información distribuida en las conexiones entre neuronas, existiendo un cierto grado de redundancia en este tipo de almacenamiento. La mayoría de los ordenadores algorítmicos y sistemas de recuperación de datos, almacenan cada pieza de información en un espacio único, localizado y direccionable. Las redes

neuronales almacenan información no localizada. Por tanto, la mayoría de las interconexiones entre los nodos de la red tendrán unos valores en función de los estímulos recibidos y se generará un patrón de salida que represente la información almacenada.

Operación en tiempo real. Una de las mayores prioridades de la mayoría de las áreas de aplicación es la necesidad de realizar grandes procesos con datos de forma muy rápida. Las redes neuronales se adaptan bien a esto debido a su implementación paralela. Para que la mayoría de las redes puedan operan en un entorno de tiempo real, la necesidad de cambio en los pesos de las conexiones o entrenamiento es mínima. Por tanto, de todos los métodos posibles, las redes neuronales son la mejor alternativa para reconocimiento y clasificación de patrones en tiempo real.

Fácil inserción dentro de la tecnología existente. Una red individual puede ser entrenada para desarrollar una única y bien definida tarea (tareas complejas, que hagan múltiples selecciones de patrones, requerirán sistemas de redes interconectadas). Debido a que una red puede ser rápidamente entrenada, testeada, verificada y trasladada a una implementación hardware de bajo coste, es fácil insertar redes neuronales para aplicaciones específicas dentro de sistemas existentes. De esta manera, las redes neuronales se pueden utilizar para mejorar sistemas de forma incremental, y cada paso puede ser evaluado antes de acometer un desarrollo más amplio.

Funcionamiento de una red de Hopfield. Una de las características del modelo de Hopfield es que se trata de una red auto-asociativa. Así, varias informaciones (patrones) diferentes pueden ser almacenadas en la red, como si de una memoria se tratase, durante la etapa de aprendizaje. Posteriormente, si se presenta a la entrada alguna de las informaciones almacenadas, la red evoluciona hasta estabilizarse, ofreciendo entonces en la salida la información almacenada que

coincide con la presentada en la entrada. Si, por el contrario, la información de entrada no coincide con ninguna de las almacenadas, por estar distorsionada o incompleta, la red evoluciona generando como salida la más parecida.

La información que recibe esta red debe haber sido previamente codificada y representada en forma de vector (como una configuración binaria si la red es discreta y como conjunto de valores reales si es contínua) con tantas componentes como neuronas (N) tenga la red.

Esa información es aplicada directamente a la única capa de que consta la red, siendo recibida por las neuronas de dicha capa (cada neurona recibe una parte de la información, un elemento del vector que representa dicha información). Si se considera, en principio, el caso de una neurona concreta de la red, esta neurona recibiría como entrada las salidas de cada una de las otras neuronas, valores que inicialmente coincidirán con los de entrada, multiplicadas por los pesos de las conexiones correspondientes. La suma de todos estos valores constituirá el valor de entrada neta de la neurona, al que le será aplicado la función de transferencia, obteniéndose el valor de salida correspondiente, 0/1 ó -1/+1 si la red es discreta y un número real en el rango [0,1] ó [-1,+1] si es contínua.

La descripción anterior correspondería a un primer paso en el procesamiento realizado por la red. Este proceso continúa hasta que las salidas de las neuronas se estabilizan, lo cual ocurrirá cuando dejen de cambiar de valor. Entonces, el conjunto de estos (N) valores de salida de todas las neuronas constituye la información de salida que ha generado la red, que se corresponderá con alguna de las informaciones que durante la etapa de aprendizaje fueron almacenadas en la misma.

Este funcionamiento puede expresarse matemáticamente de la siguiente forma:

1. En el instante inicial (t=0) se aplica la información de entrada (Valores e_1, e_2,.., e_N).

$$S_i\ (t=0) = e_i \qquad 1 \le i \le N$$

Inicialmente la salida de las neuronas coincide con la información aplicada a la entrada.

2. La red realiza iteraciones hasta alcanzar la convergencia (hasta que s_i (t+1) sea igual a s_i (t)).

$$S_i(\mathrm{t}+1) = f\left(\sum_{j=1}^{N} W_{ij} S_j(t) \; - \theta_i\right) \quad 1 \le i \le N$$

Donde f es la función de transferencia (activación) de las neuronas de la red. En el caso del modelo discreto, si se trabaja con valores binarios -1 y +1, la salida se obtendría según la función escalón:

$$S_i(t+1) = \begin{cases} +1 & \sum_{j=1}^{N} W_{ij} S_j(t) \; > \; \theta_i \\ Si(t) & \sum_{j=1}^{N} W_{ij} S_j(t) \; = \; \theta_i \\ -1 & \sum_{j=1}^{N} W_{ij} S_j(t) \; > \; \theta_i \end{cases}$$

El proceso se repite hasta que las salidas de las neuronas permanecen sin cambios durante algunas iteraciones. En ese instante la salida (s_1, s_2,..,s_N) representa la información almacenada por la red que más se parece a la información presentada en la entrada (e_1, e_2,..,e_N).

El proceso se repite hasta que las salidas de las neuronas permanecen sin cambios durante algunas iteraciones. En ese instante la salida (s_1, s_2,..,s_N) representa la

información "almacenada" por la red que más se parece a la información presentada en la entrada (e_1, e_2,..,e_N).

El funcionamiento descrito corresponde a una red Hopfield Discreta clásica, ya que trabaja con informaciones binarias. Sin embargo, existe una variación del modelo desarrollada también por Hopfield que pretende parecerse un poco más al funcionamiento de las neuronas reales, se trata de la Red Hopfield Continúa, con funciones de activación de las neuronas de tipo sigmoidal, que ofrece más posibilidades que la anterior, ya que permite almacenar patrones formados por componentes no binarios, sino valores reales (por ejemplo imágenes en color o en blanco y negro con diferentes tonalidades de grises) y además permitirá resolver determinados problemas generales de optimización.

Tanto en este caso como en el de la red discreta, se pueden distinguir diferentes versiones del modelo en función de la forma temporal en que se lleva a cabo la generación o actualización de las salidas de las neuronas de la red. Si esta actualización se realiza de forma simultánea en todas las neuronas se trata de una Red Hopfield con funcionamiento paralelo o síncrono, ya que supone que todas las neuronas son capaces de operar sincronizadas y que, por tanto, la salida es generada al mismo tiempo por todas ellas en cada iteración, de tal forma que en la próxima iteración (t+1) todas van a utilizar como entradas las salidas generadas por las otras en el instante anterior (t). Si, por el contrario, las neuronas trabajan de forma secuencial, actualizándose sólo la salida de una neurona en cada iteración, se tratará de una Red Hopfield con funcionamiento secuencial o asíncrono. En este caso, ocurre que la salida a la que converge la red puede ser diferente en función del orden de la secuencia de activación de las neuronas.

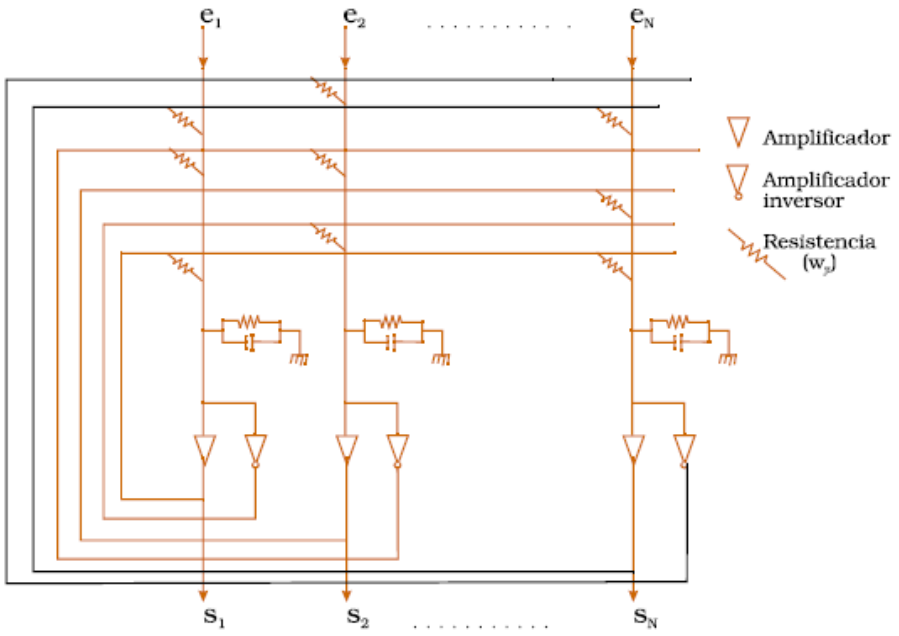
También existe la posibilidad, sólo utilizada en las redes Hopfield Contínuas, que trabajan con valores no binarios, de que la generación de la salida de todas las neuronas se realice de forma simultánea (síncrona) y continuada en el tiempo. En

este caso, el funcionamiento de la red tendría que ser representado en forma de una ecuación diferencial:

$$\tau_i \frac{ds_i}{dt} = f(\sum_{j=1}^{N} W_{ij} S_j - \theta_i) - S_i$$

Donde f es la función de activación de las neuronas, que será de tipo sigmoidal, y τ_i es un parámetro, denominado tasa de retardo, que pretende representar una característica biológica de las neuronas como es el retraso que se produce en la recepción, por parte de una neurona i, de los valores de las salidas generados por las otras neuronas a las que está conectada.

Figura 94.Implementación física de una red Hopfield continúa



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. REDES NEURONALES ARTIFICIALES. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.188.

Si se pretende simular una red Hopfield Contínua como la anterior, puede hacerse resolviendo la ecuación diferencial utilizando métodos numéricos, aunque también, y sobre todo, puede implementarse físicamente de forma directa mediante un circuito analógico que tenga un comportamiento gobernado por una ecuación similar (Figura 94). En estos circuitos los pesos entre neuronas se implementan como resistencias, las funciones de activación mediante amplificadores operacionales y la tasa de retardo de cada neurona se consigue colocando, en paralelo, una resistencia y un condensador a la entrada del amplificador correspondiente.

Aplicaciones de la red de Hopfield. A pesar de las limitaciones comentadas en el apartado anterior, el modelo de Hopfield ha sido uno de los más utilizados, como se ha descrito o con algunas modificaciones, siendo esto debido principalmente a la relativa facilidad de implementación física de esta red utilizando tecnología VLSI (muy alta escala de integración).
En cuanto a las aplicaciones más conocidas del modelo [Simpson 90], destacan las relacionadas con el reconocimiento de imágenes y de voz, el control de motores y, sobre todo, la resolución de problemas de optimización. En este último ámbito se ha aplicado para la resolución de ecuaciones y del problema del viajante, manipulación de grafos, procesado de señales (por ejemplo el diseño de conversores Analógico-Digitales) y de imágenes, reconocimiento de patrones y control de motores.

Resolución de problemas de optimización. Una de las aplicaciones de las redes neuronales en general y del modelo de Hopfield en particular es la resolución de problemas de optimización complejos, aquellos cuyo tiempo de resolución mediante algoritmos convencionales crece exponencialmente (k^N) o factorialmente (N!) con el aumento del tamaño (N) del problema (por ejemplo, el problema clásico de colocar N reinas en un tablero de ajedrez de NxN cuadros sin que se den jaque, que trataría de encontrar una posición de las piezas de entre las $N^2!$ (N^2-N)!/N! combinaciones diferentes, que en caso de un tablero real de N=8 serían más de cuatro mil millones

de combinaciones, para este problema pueden encontrarse algoritmos convencionales con tiempo de resolución proporcionales a N!). En realidad, cuando las redes se aplican para tal función, lo que se está haciendo es intentar resolver el problema mediante un algoritmo paralelo formulado en términos de una red neuronal, que no es sino un sistema de computación paralela, al estar trabajando multitud de diferentes elementos de proceso (neuronas) simultáneamente.

La utilización del modelo de Hopfield para la resolución de problemas de optimización se basa en la idea de intentar fijar el objetivo del problema mediante una expresión matemática, denominada función de coste o función objetivo, que haya que minimizar. A continuación se compara con la expresión general de la función de energía de una Red Hopfield, determinándose los valores de los pesos (w_{ij}) y de los umbrales (θ_i) en términos de parámetros de la función de objetivo para que ambas expresiones sean equivalentes. De esta forma, cuando se pone en funcionamiento una Red Hopfield Contínua con los valores calculados, ésta itera hasta alcanzar un mínimo de la función energía que, en este caso, coincidirá con un mínimo de la función objetivo, encontrando así una posible solución de mínimo coste del problema de optimización. Generalmente no es fácil encontrar el mínimo global debido a presencia de mínimos locales de los que habrá que escapar realizando un complicado ajuste de parámetros constantes de la función de coste y de las ganancias (pendientes) de las funciones de activación de las neuronas de la red.
Aunque en gran medida esta organización neuronal está predeterminada genéticamente, es probable que parte de ella se origine mediante el aprendizaje. Esto sugiere, por tanto, que el cerebro podría poseer la capacidad inherente de formar mapas topológicos de las informaciones recibidas del exterior. De hecho, esta teoría podría explicar su poder de operar con elementos semánticos: algunas áreas del cerebro simplemente podrían crear y ordenar neuronas especializadas o grupos con características de alto nivel y sus combinaciones. Se trataría, en definitiva, de construir mapas espaciales para atributos y características.

El modelo Kohonen. En base a estas ideas, T. Kohonen presentó en 1982 un sistema con un comportamiento semejante. Se trataba de un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro. El objetivo de Kohonen era demostrar que un estímulo externo (información de entrada) por sí solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas.

Este modelo tiene dos variantes denominadas LVQ (Learning Vector Quantization) y TPM (Topology-Preserving Map) o SOM (Self-Organizing Map). Ambas se basan en el principio de formación de mapas topológicos para establecer características comunes entre las informaciones (vectores) de entrada a la red, aunque difieren en las dimensiones de éstos, siendo de una sola dimensión en el caso de LVQ y bidimensional, e incluso tridimensional, en la red TPM.

Aprendizaje. El aprendizaje en el modelo de Kohonen es de tipo *Off Line*, por lo que se distingue una etapa de aprendizaje y otra de funcionamiento. En la etapa de aprendizaje se fijan los valores de los pesos de las conexiones (feedfoward) entre la capa de entrada y la de salida.

Esta red utiliza un aprendizaje no supervisado de tipo competitivo. Las neuronas de la capa de salida compiten por activarse y sólo una de ellas permanece activa ante una determinada información de entrada a la red. Los pesos de las conexiones se ajustan en función de la neurona que haya resultado vencedora.

Durante la etapa de entrenamiento se presenta a la red un conjunto de informaciones de entrada ("vectores de entrenamiento") para que ésta establezca, en función de la semejanza entre los datos, las diferentes categorías (una por neurona de salida) que servirán durante la fase de funcionamiento para realizar clasificaciones de nuevos datos que se presenten a la red. Los valores finales de

los pesos de las conexiones entre cada neurona de la capa de salida con las de entrada se corresponderán con los valores de los componentes del vector de aprendizaje que consigue activar la neurona correspondiente. En el caso de existir más patrones de entrenamiento que neuronas de salida, más de uno deberá asociarse con la misma neurona, es decir, pertenecerán a la misma clase. En tal caso los pesos se obtienen como un promedio de dichos patrones.

En este modelo, el aprendizaje no concluye después de presentarle una vez todos los patrones de entrada, sino que habrá que repetir el proceso varias veces para refinar el mapa topológico de salida, de tal forma que cuantas más veces se presenten los datos, tanto más se reducirán las zonas de neuronas que se deben activar ante entradas parecidas, consiguiendo que la red pueda realizar una clasificación más selectiva.

El algoritmo de aprendizaje utilizado para establecer los valores de los pesos de las conexiones entre las N neuronas de entrada y las M de salida es el siguiente:

1. En primer lugar se inicializan los pesos (wji) con valores aleatorios pequeños y se fija la zona inicial de vecindad entre las neuronas de salida.

2. A continuación se presenta a la red una información de entrada (la que debe "aprender") en forma de vector $E_k = (e_1^{(k)}, \dots, e_N^{(k)})$ cuyas componentes $e_1^{(k)}$serán valores contínuos.

3. Puesto que se trata de un aprendizaje competitivo, se determina la neurona vencedora de la capa de salida. Esta será aquella j cuyo vector de pesos W_j(vector cuyas componentes son los valores de los pesos de las conexiones entre esa neurona y cada una de las neuronas de la capa de entrada) sea el más parecido a la información de entrada E_k (patrón o vector de entrada).

Para ello se calculan las distancias o diferencias entre ambos vectores considerando una por una todas las neuronas de salida. Suele utilizarse la distancia euclídea o la siguiente expresión, que es similar a aquella pero eliminando la raíz cuadrada:

$$d_j = \sum_{i=1}^{N} \left(e_i^{(k)} - w_{ij} \right)^2 \qquad 1 \leq j \leq M$$

Siendo:

$e_i^{(k)}$: Componente i-ésimo del vector k-ésimo de entrada.

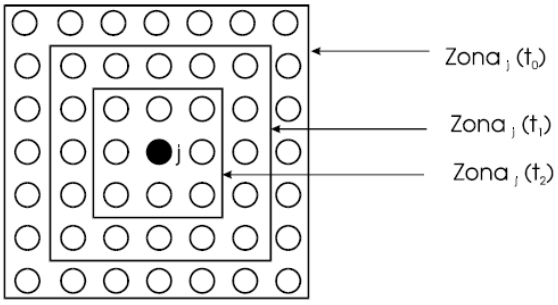
W_j: Peso de la conexión entre la neurona i de la capa de entrada y la neurona j de la capa de salida.

4. Una vez localizada la neurona vencedora (j*) se actualizan los pesos de las conexiones entre las neuronas de entrada y dicha neurona, así como los de las conexiones entre las de entrada y las neuronas vecinas de la vencedora. En realidad, lo que se consigue con esto es asociar la información de entrada con una cierta zona de la capa de salida.

$$w_{ji}(t+1) = w_{ji}(t) + \alpha(t)\left[e_i^{(k)} - w_{j*i}(t) \right] \; para\, j \in Zona_{j*}(t)$$

$Zona_{j*}(t)$ Es la zona de vecindad alrededor de la neurona vencedora j* en la que se encuentran las neuronas cuyos pesos son actualizados. El tamaño de esta zona se puede reducir en cada iteración del proceso de ajuste de los pesos, con lo que el conjunto de neuronas que pueden considerarse vecinas cada vez es menor (Figura 86). Sin embargo, en la práctica es habitual considerar una zona fija en todo el proceso de entrenamiento de la red.

Figura 95. Posible evolución de la zona de vecindad



Fuente: HILERA, José Ramón y MARTINEZ, Víctor José. Sobre antecedentes de las Redes neuronales. REDES NEURONALES ARTIFICIALES. Fundamentos, modelos y aplicaciones. Madrid: RA-MA Editorial, 1995. p.261.

El término $\alpha(t)$ es un parámetro de ganancia o coeficiente de aprendizaje, con un valor entre 0 y 1, decrece con el número de iteraciones (t) del proceso de entrenamiento. De tal forma que cuando se ha presentado un gran número de veces todo el juego de patrones de aprendizaje (500≤ t ≤ 10000) su valor es prácticamente nulo, con lo que la modificación de los pesos es insignificante.

Suele utilizarse alguna de las siguientes expresiones:

$$\alpha(t) = \frac{1}{t} \qquad \alpha(t) = \alpha_1\left(1 - \frac{t}{\alpha_2}\right)$$

Siendo α_1 un valor de 0.1 ó 0.2 y α2 un valor próximo al número total de iteraciones del aprendizaje. Suele tomarse un valor α_2=10000.

5. El proceso se debe repetir, volviendo a presentar todo el juego de patrones de aprendizaje $\mathrm{E}_1, \mathrm{E}_2, \ldots$, un mínimo de 500 veces (t ≥ 500).

Aplicaciones. El modelo de Kohonen es uno de los más útiles en computación neuronal, a pesar de sus limitaciones en cuanto a la duración del proceso de aprendizaje y a la imposibilidad de aprender nuevos datos sin tener que volver a repetir completamente el proceso de aprendizaje con todos los patrones. Esta utilidad se debe a su capacidad para establecer clases o categorías de datos sin supervisión.

Como aplicaciones destacan las relacionadas con el reconocimiento de patrones (voz, texto, imágenes, señales, etc.), codificación de datos, compresión de imágenes y resolución de problemas de optimización, como el del Viajante. También se ha utilizado este modelo en robótica, comprobándose su utilidad en el diseño de sistemas para controlar el movimiento de un brazo mecánico en un espacio tridimensional. En dichos sistemas, se utiliza una red de Kohonen para aprender las magnitudes tensoriales necesarias para moverse en un entorno real, considerando los efectos del desgaste que pueden alterar la dinámica del brazo con el transcurso del tiempo.

En los siguientes apartados se describen algunas de estas aplicaciones, concluyendo con la presentación de un nuevo tipo de red neuronal denominada Counter-propagation, que puede considerarse también como una aplicación del modelo de Kohonen para combinar aprendizaje no supervisado y no supervisado en un misma red del tipo *Perceptron* multicapa que pueda servir de alternativa al conocido algoritmo supervisado Back-propagation.

Anexo D. IMPLEMENTACIÓN, DESCRIPCIÓN Y FUNCIONAMIENTO DEL SOFTWARE

IMPLEMENTACIÓN DEL SOFTWARE.

El presente software “NEURAL NET OPTRASOFT 2.0”, es la actualización y adaptación de software creado en el proyecto “APLICACIÓN DE REDES NEURONALES PARA LA OPTIMIZACION DE LA TRAYECTORIA DE LA HERRAMIENTA EN EL FRESADO DE CAVIDADES COMPLEJAS”[6] el cual permite la obtención de una trayectoria para el fresado de una cavidad, aplicando algoritmos y estructuras computacionales basados en aprendizaje y comparación de parámetros, no obstante se mantiene el mismo estilo y método de recolección de datos, visualización del proceso en todo momento por medio de las pantallas, lo que facilita una fácil interacción entre el usuario y el programa.

Las actualizaciones realizadas son la implementación de red neuronal de Hopfield, y el método de programación lineal entera, y unas mejoras hechas al método de Kohonen que lo hacen más eficiente, aplicadas a la solución del problema de TSP (traveling salesman problem), generando unas trayectorias alternativas al método ya implementado. La posibilidad de mover la matriz de puntos con el objeto de ajustar los puntos de la herramienta a la figura.

DESCRIPCIÓN DEL SOFTWARE

El software NEURAL NET OPTRASOFT Versión 2.0 utiliza una interfaz que presenta una Pantalla de Presentación Splash, una pantalla para cada uno de los

---

6 GALVIZ PARRA, Edgar y GONZALEZ ALMEIDA, Fabio Leonel. Aplicación de redes neuronales para la optimización de la trayectoria de la herramienta en el fresado de cavidades complejas. Bucaramanga, 2012, Trabajo de grado (Ingeniero Mecánico). Universidad Industrial de Santander. Facultad de ingenierías Físico mecánicas. Disponible en el catálogo en línea de la Biblioteca de la Universidad Industrial de Santander: <http:// http://tangara.uis.edu.co/>

cuatro pasos del proceso, y dentro de cada paso su respectivo subproceso, los resultados dan una imagen en 3 dimensiones y un archivo con los puntos de la trayectoria.

Pantalla de presentación (SPLASH). Esta pantalla (ver Figura 96) es la presentación del software y se muestra una vez se da inicio al programa, esta imagen permanece por un periodo aproximado de tres (3) segundos una vez pasado este tiempo se cierra y da paso al step 1

Figura 96. Pantalla de presentación (Splash)

Pantalla Step 1. En esta pantalla se realiza la adquisición y configuración de los datos o parámetros iniciales para el mecanizado, y de igual forma se carga la imagen de la cavidad a analizar. (Ver Figura 97).

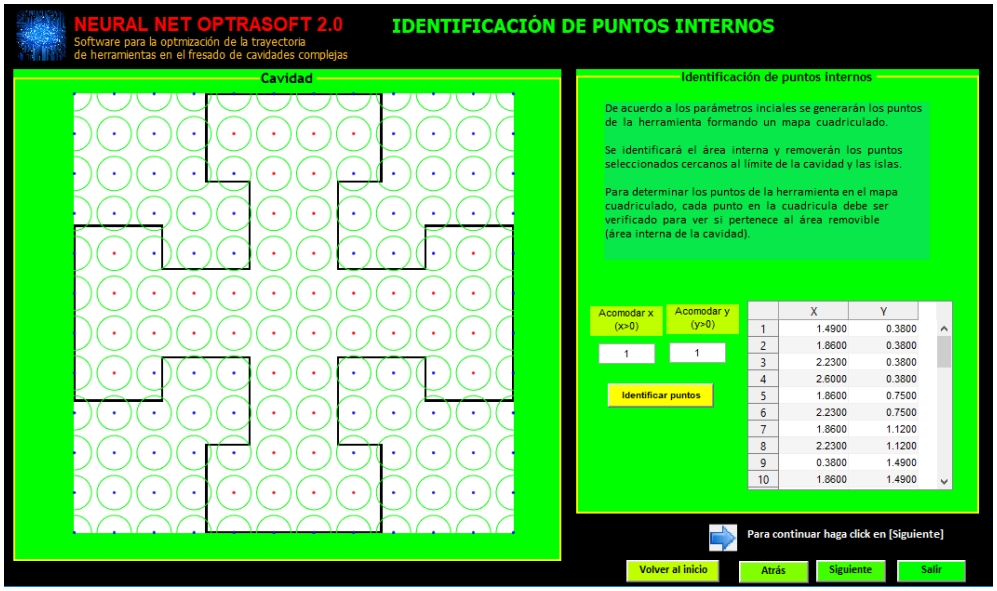
Figura 97. Pantalla Step 1.

Pantalla Step 2. En esta pantalla de acuerdo a los parámetros iniciales establecidos en el Step 1, se generarán los puntos de la herramienta formando un mapa cuadriculado, llegado al caso se puede reubicar los puntos iniciales del mapa para ajustarlo a la imagen.

Se identificará el área interna y removerán los puntos seleccionados cercanos al límite de la cavidad y las islas.

Para determinar los puntos de la herramienta en el mapa cuadriculado, cada punto en la cuadricula debe ser verificado para ver si pertenece al área removible, interna de la cavidad (Ver Figura 98).

Figura 98. Pantalla Step 2



Pantalla Step 3. De acuerdo con los puntos hallados en el Step 2, que corresponden a la cavidad se tomara para resolver el problema del TSP (Travelling Salesman Problem) uno de los 3 métodos (SOM (Self-Organizing Maps), Red Hopfield, Programación lineal entera).

El método SOM presentado inicialmente comienza por un anillo inicial (banda elástica) compuesto de un sistema de nodos en el plano, seguida por una evolución iterativa tal que todos los puntos de ciudades (equivalente a ciudades en el algoritmo teórico) son dibujados dentro del cuadrilátero (Ver figura 99).

Si todos los puntos son dibujados dentro del cuadrilátero, la conexión de estos puntos produce una solución óptima al TSP

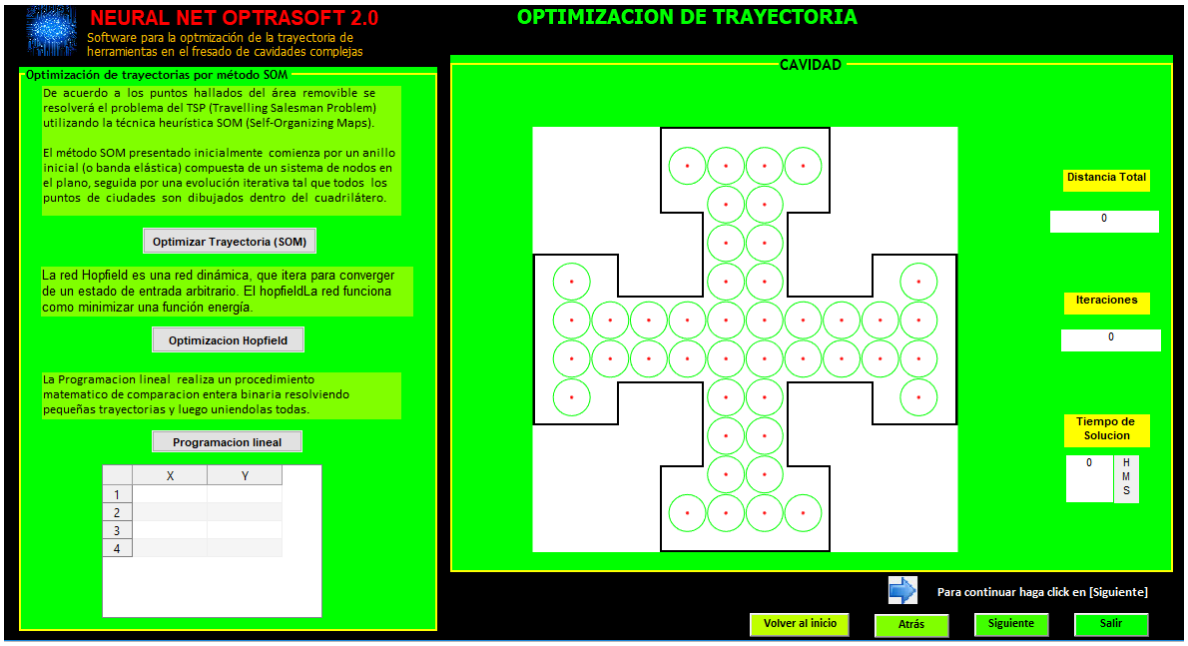
La Red Hopfield tomara los puntos, seguidamente realizara una matriz de distancias, la cual usara para identificar una ruta inicial y de ahí por medio de minimización de energía hallar una trayectoria optima, seguidamente se corregirán cruces entre rutas.

El método de Programación Lineal Entera toma los puntos, realiza una matriz de distancias, y por medio de evaluación de limites busca trayectorias (tures) más

pequeños que el general, posteriormente realiza iteraciones para unir todos los tours en uno solo.

Una vez realizadas las iteraciones y establecer la trayectoria final por cada uno de los métodos, se muestra la distancia total de la trayectoria, la cantidad de iteraciones y el tiempo de solución.

Figura 99. Pantalla Step 3



Pantalla Step 4. En este Step se presiona la ruta de la trayectoria y se corrige un poco la distancia del Step anterior, debido a saltos sobre el maquinado o trayectorias cruzadas. (Ver figura 100)

Figura 100. Pantalla Step 4.

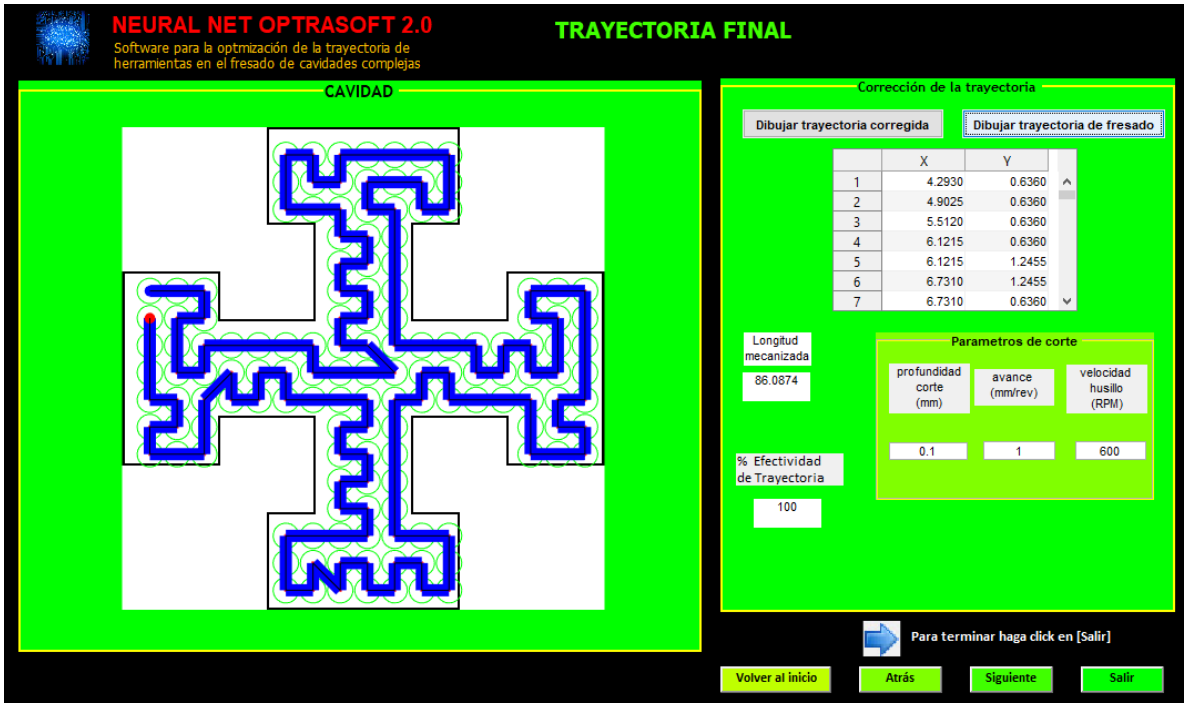
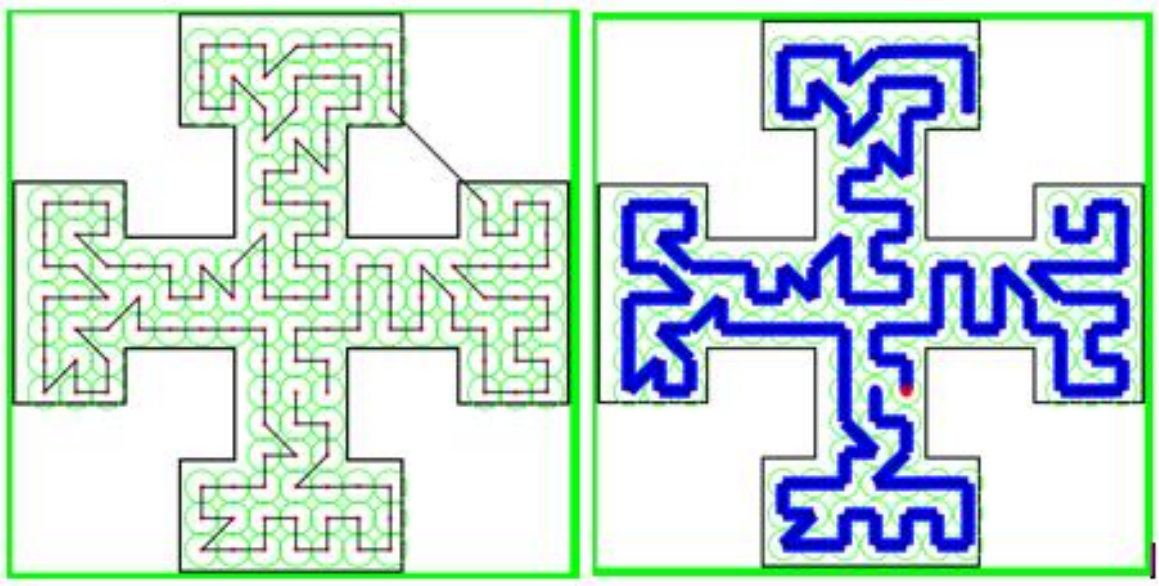


Figura 101. Corrección de trayectoria fuera de la cavidad



Pantallas finales. Una vez realizado los pasos anteriores se muestra una imagen tridimensional la cual represnta la profundidad de corte de cada ruta. Cuya

profundidad de corte puede ser modificada de acuerdo al tipo de mecanizado que se realice en la pantalla del step 4 (ver figura 102)

Figura 102. Profundidad de corte

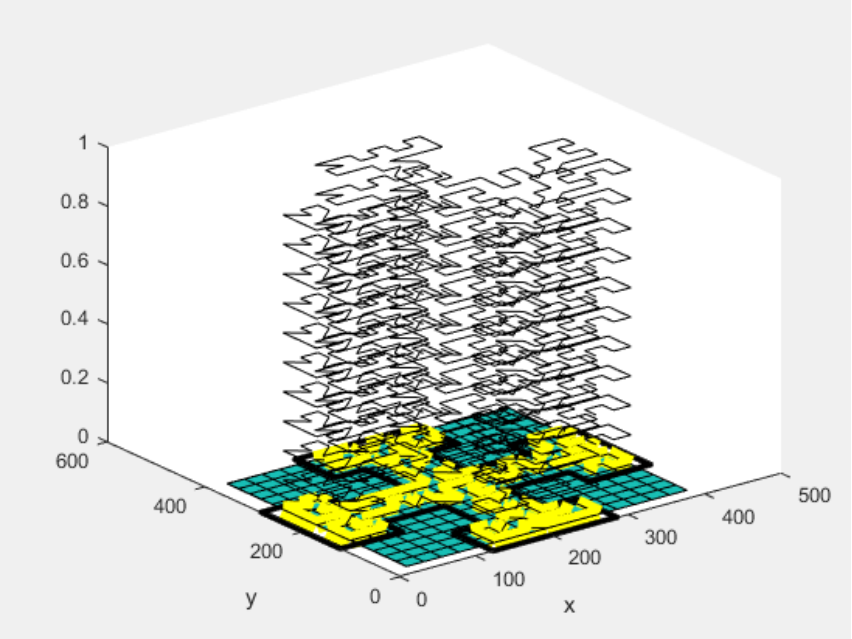


Figura 103. Coordenadas de puntos maquinales

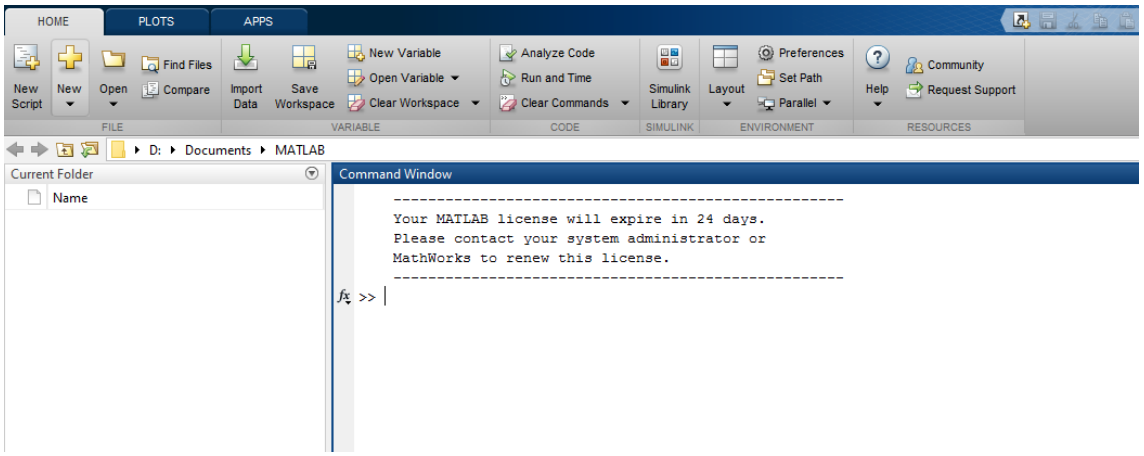
277.00	254.00
254.00	231.00
231.00	231.00
208.00	185.00
185.00	208.00
231.00	231.00
208.00	185.00
208.00	231.00
231.00	254.00
277.00	277.00
276.56	254.00
254.00	231.00
231.00	208.00
208.00	208.00
185.00	185.00
185.00	185.00
161.83	139.00
139.00	162.00
162.00	139.00
70.00	47.00
24.00	24.00
24.00	24.66
47.00	24.00
24.00	24.00
47.00	70.00
70.00	47.00
70.00	93.00
70.00	47.00
47.00	47.00

FUNCIONAMIENTO DEL SOFTWARE

NEURAL NET OPTRASOFT 2.0 está desarrollado en MATLAB y permite la interacción a través de interfaces gráficas de usuario, por su facilidad, secuencia de interfaces, y la accesibilidad que tienen los estudiantes de la Universidad Industrial de Santander a este Software de manera gratuita y licenciado, por esto el estudiante esta familiarizado a esta herramienta que guia al usuario a la procesamiento de las cavidades.

Iniciar el programa.
Para ejecutar NEURAL NET OPTRASOFT 2.0, es necesario primero abrir el entorno de MATLAB (Ver figura 104):

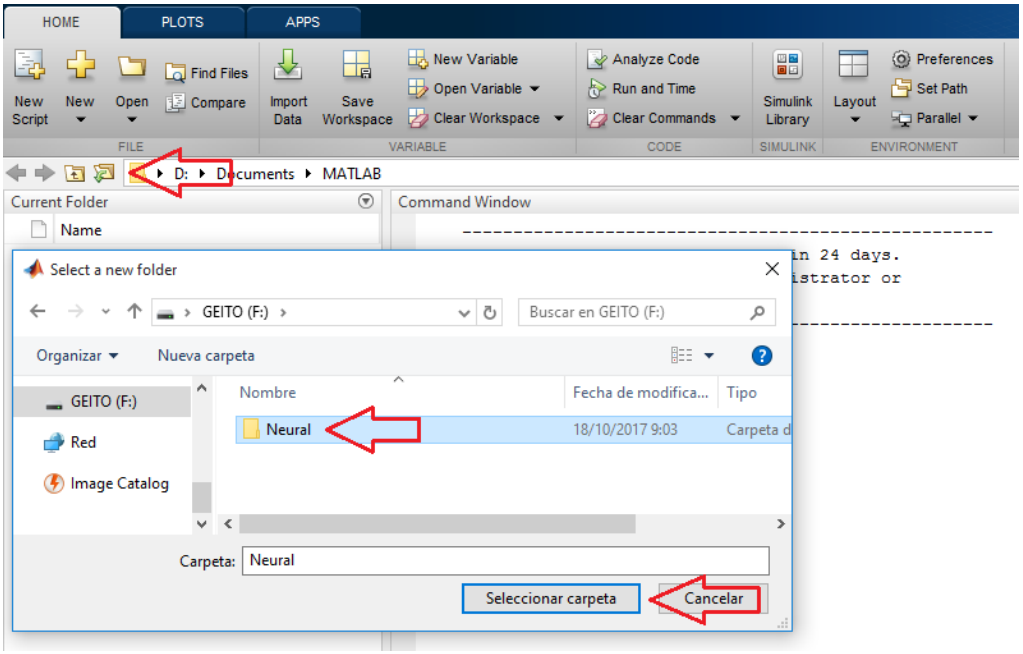
Figura 104. Pantalla de entorno MATLAB



Luego se procede a cargar los archivos o carpetas concernientes al desarrollo del software para que MATLAB lo ejecute, en la dirección en la cual se encuentren, se

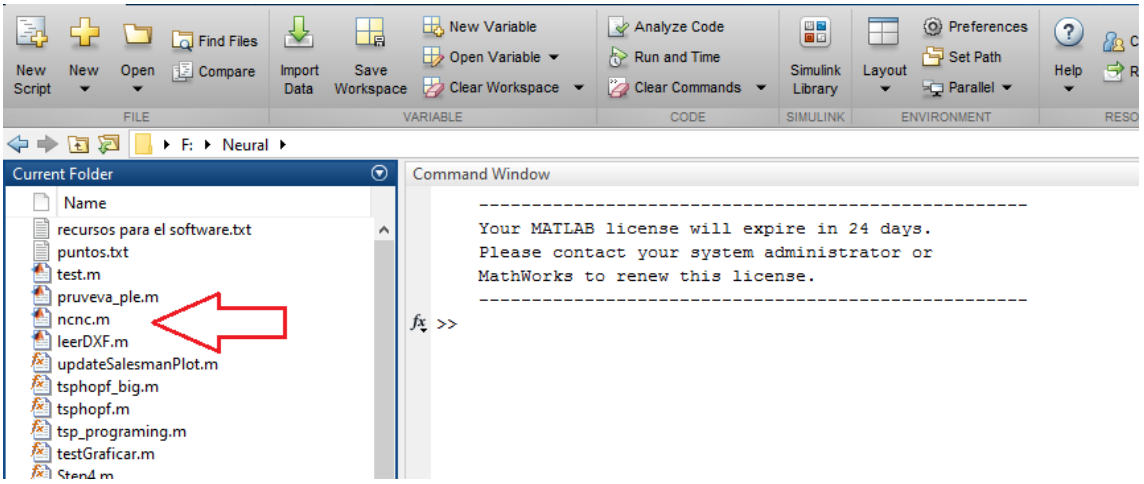
abrirá un cuadro de búsqueda en el cual se debe identificar la carpeta portadora del software (Ver figura 105)

Figura 105. Cargar carpeta



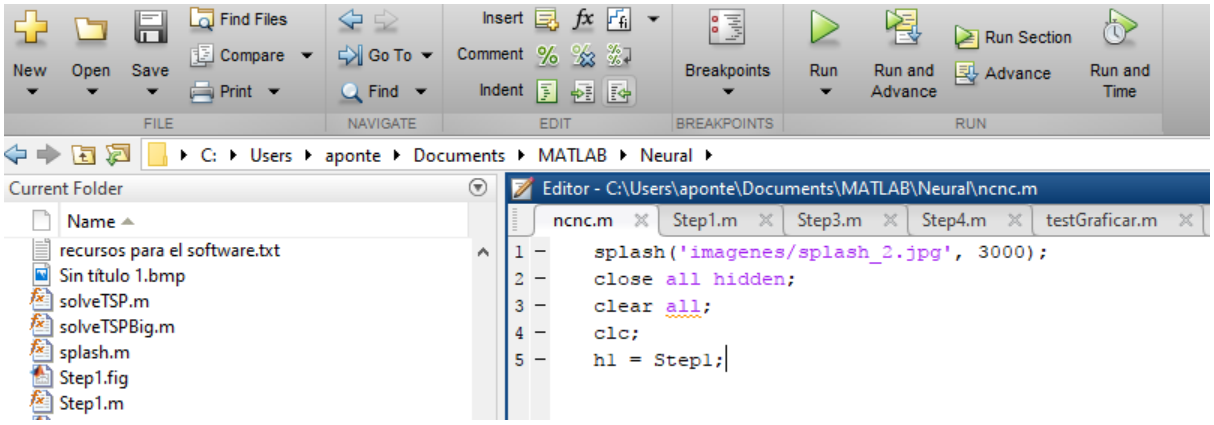
Después de seleccionar la carpeta portadora se procede a ubicar en el panel lateral izquierdo el archivo que tiene por nombre “ncnc.m” que me ejecuta el NEURAL NET OPTRASOFT 2.0 (Ver Figura 106)

Figura 106. Ejecución del programa



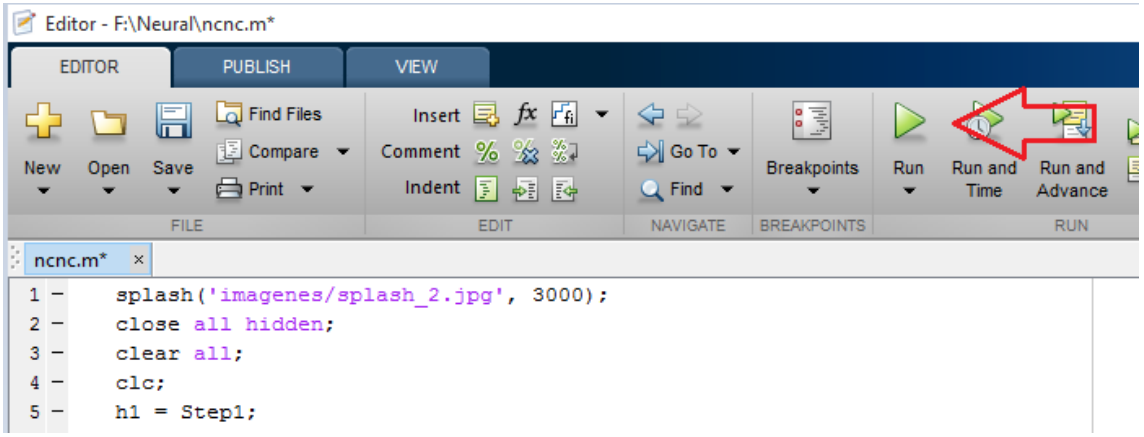
Seguidamente se abre un editor el cual me carga la programación en MATLAB del software NEURAL NET OPTRASOFT 2.0 (ver figura 107)

Figura 107. Pantalla de inicio de programa



Posterior a cargar la programacion se procede a reproducir el software NEURAL NET OPTRASOFT 2.0 en el botón “Run” (ver Figura 108)

Figura 108. Correr el código



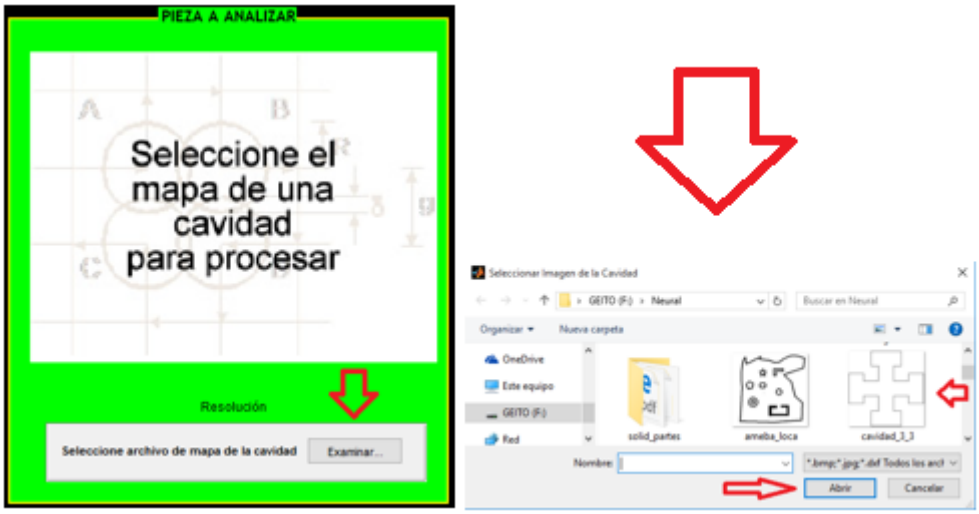
Se ejecutara el código y empezara a funcionar el software abriendo en seguida la pantalla de bienvenida al programa (Figura 96) y posteriormente un cuadro de dialogo del Step 1 (Figura 97) en el cual se puede apreciar:

Botones especiales: Estas funciones permiten volver a iniciar el programa, Ir al paso anterior, Ir al paso siguiente y Salir del programa, respectivamente (Ver figura 109) y la Selección de la cavidad a procesar junto a parámetros de la herramienta.

Figura 109. Botones especiales

En La Selección de la cavidad o pieza a procesar se puede observar el boton examinar en la parte inferior del cuadro “pieza a analizar” el cual direcciona a un cuadro de búsqueda que permite seleccionar la figura a mecanizar (ver figura 110)

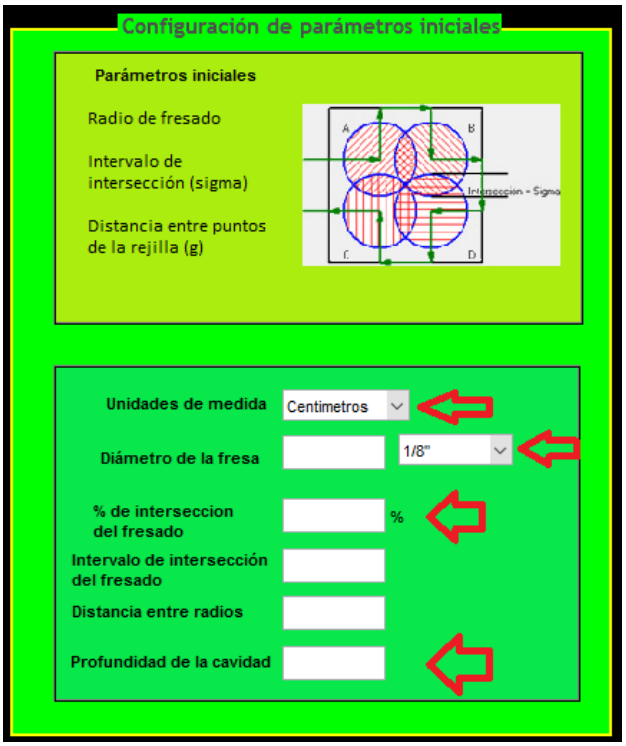
Figura 110. Selección de cavidad



Una vez seleccionada y abierta la figura a procesar se ingresan los datos de entrada iniciales los cuales permiten elegir unidades a utilizar en la simulación aunque internamente se procesan los datos en píxeles; las unidades posibles son Pulgadas [in], Centímetros [Cm] y Píxeles, una vez seleccionadas automáticamente se capturan los datos en los cajones de texto del radio de la fresa y se debe introducir

el valor del porcentaje de intersección, inmediatamente el software identifica el valor del intervalo de intersección y el de la distancia entre radios de la rejilla, posteriormente debe ingresarse el valor de la profundidad de la cavidad (ver figura 111)

Figura 111. Configuración de parámetros iniciales

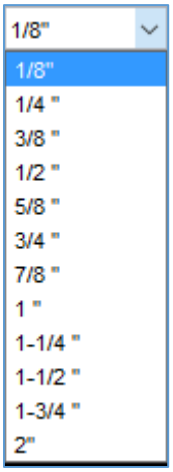


Unidades de medida: Se escogen las unidades de medida para la simulación. (Ver figura 112).

Figura 112. Selección de unidades de medida

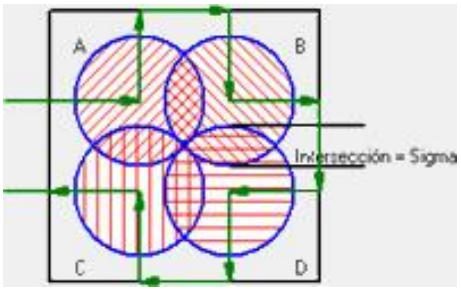
Diámetro de la fresa: Escoge el Diámetro de la herramienta en el recuadro de opciones, estandarizada con las fresas comúnmente usadas en la industria: 1/8, ¼, 3/8, ½, 5/8, ¾, 7/8, 1, 1¼, 1½, 1-3/4, y 2 pulgadas (Ver figura 113).

Figura 113. Selección de diámetro de fresa



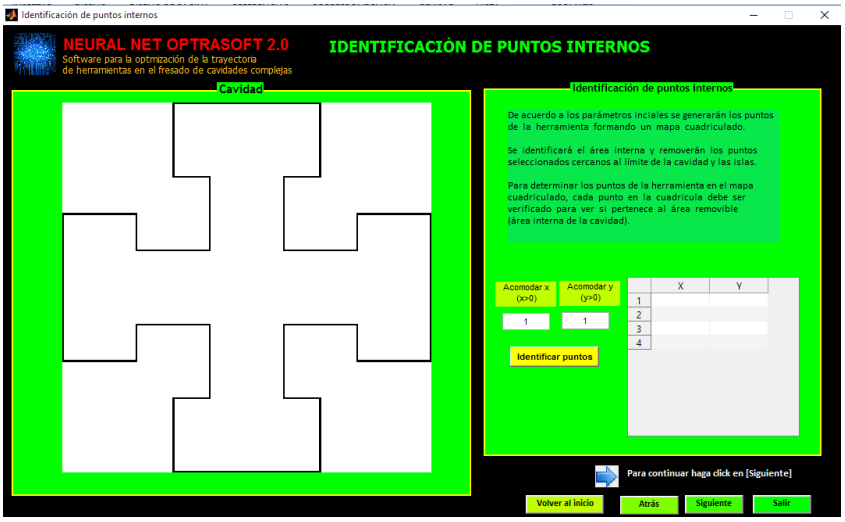
Porcentaje de intersección del fresado: Es el porcentaje de superposición entre dos radios de fresa, y da el parámetro para calcular el intervalo y distancia entre centros.
Intervalo de intersección del fresado: Es la distancia de superposición entre dos radios de fresa, debe ser menor que el doble del radio de la fresa.
Distancia entre centros: Es la distancia a la que se asignarán los puntos en la rejilla de maquinado que se detallará en pasos posteriores, definido por el porcentaje de intersección (Ver figura 114).
Profundidad de la cavidad: Distancia en profundidad a la que se hará el fresado de la pieza.

Figura 114. Parámetros de la fresa



Una vez definidos todos los valores anteriores se continúa con la secuencia haciendo clic en el botón Siguiente.

Figura 115.Pantalla de identificación de puntos internos



Creación de la rejilla uniforme de puntos. Una vez cargados los datos iniciales y la imagen, se procede a calcular los puntos de rejilla de la cavidad, haciendo clic en el botón “Identificar puntos” (ver figura 115), en el caso de que la imagen no quede bien centrada se puede centralizar con la herramienta acomodar “x” o acomodar “y” (Ver figura 107), una vez seleccionada la opción de identificar puntos el software los identifica en la figura tabulada situada en la parte lateral inferior derecha (Figura 116)

Figura 116. Centralizar la imagen

Figura 117. Generación de puntos

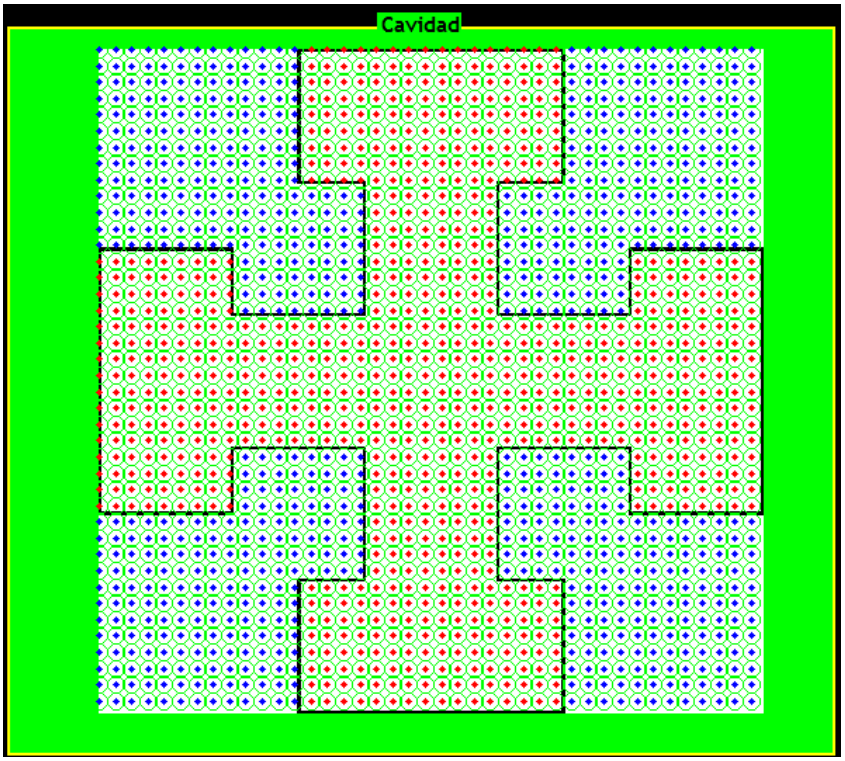


Figura 118. Puntos tabulados

	X	Y
1	1.3100	0.0100
2	1.4100	0.0100
3	1.5100	0.0100
4	1.6100	0.0100
5	1.7100	0.0100
6	1.8100	0.0100
7	1.9100	0.0100
8	2.0100	0.0100
9	2.1100	0.0100
10	2.2100	0.0100

Estos resultados se interpretan de acuerdo a las siguientes convenciones:

Punto que pertenece al área removible en rojo (maquinable)

Punto que pertenece al área no removible en azul (interior de las islas, bordes de la cavidad, o área externa a la cavidad).

Anexo E. CÓDIGOS DEL SOFTWARE NEURALNET 2.0

Códigos step 1

```
function varargout = Step1(varargin)
% STEP1 MATLAB code for Step1.fig
%      STEP1, by itself, creates a new STEP1 or raises the existing
%      singleton*.
%
%      H = STEP1 returns the handle to a new STEP1 or the handle to
%      the existing singleton*.
%
%      STEP1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STEP1.M with the given input arguments.
%
%      STEP1('Property','Value',...) creates a new STEP1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Step1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Step1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Step1

% Last Modified by GUIDE v2.5 22-Oct-2017 15:17:04
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Step1_OpeningFcn, ...
                   'gui_OutputFcn',  @Step1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

 % --- Executes just before Step1 is made visible.
function Step1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Step1 (see VARARGIN)

% Choose default command line output for Step1
handles.output = hObject;
scrsz = get(0,'ScreenSize');
```

```
pos_act = get(gcf, 'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

% Update handles structure
axes(handles.axes1);
img = imread('imagenes/red_neural.jpg');
imshow(img);
axes(handles.axes4);
fig = imread('imagenes/default.jpg');
imshow(fig);
axes(handles.axes3);
fig = imread('imagenes/next.jpg');
imshow(fig);
axes(handles.axes2);
fig = imread('imagenes/param_fresa.jpg');
imshow(fig);
assignin('base','U_FACTOR',0.0254); %conversion base de pixeles a in/cm
set(handles.txtRadioFresa,'string','');
set(handles.txtSigma,'string','');
set(handles.txtG,'string','');
set(handles.porcentaje,'string','');
set(handles.txtProfundidad,'string','');
set(handles.popupmenu1,'value',1);

%============================
set(handles.figure1,'resize','on');
```

```
%============================

guidata(hObject, handles);
% UIWAIT makes Step1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
 % --- Outputs from this function are returned to the command line.
function varargout = Step1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbuttonBack.
function pushbuttonBack_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonBack (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbuttonNext.
function pushbuttonNext_Callback(hObject, eventdata, handles)%para pasar al
siguiente step
% hObject    handle to pushbuttonNext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
radioFresa = str2double(get(handles.txtRadioFresa,'string'));
sigma = str2double(get(handles.txtSigma,'string'));
g = str2double(get(handles.txtG,'string'));
```

```
prof = str2double(get(handles.txtProfundidad,'string'));
medida = str2double(get(handles.edit6,'string'));
imagen=evalin('base','imagen');
U_F_ESC=(size(imagen,2)*0.254)/medida;
U_FACTOR = evalin('base','U_FACTOR');
U_FACTOR=U_FACTOR/U_F_ESC;
radioFresa = fix(radioFresa/U_FACTOR);
sigma = fix(sigma/U_FACTOR);
g = fix(g/U_FACTOR);

assignin('base','radioFresa',radioFresa/2);
assignin('base','sigma',sigma);
assignin('base','g',g);
assignin('base','profundidad',prof);
assignin('base','U_FACTOR',U_FACTOR);
assignin('base','U_F_ESC',U_F_ESC);
ocultarVentana('h1');
h2 = Step2;
assignin('base','h2',h2);

% --- Executes on button press in pushbuttonExit.
function pushbuttonExit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
choice = questdlg('¿Realmente desea salir?','Salir','Si','No','No');
switch choice
    case 'Si'
    clear all;
    clc;
```

```
    close;
end

% --- Executes on button press in pushbuttonHome.
function pushbuttonHome_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonHome (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function pushbuttonExit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% image_rgb = imread('imagenes/icon.jpg');
% set(hObject,'cdata',image_rgb);

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
U_FACTOR_0 = evalin('base','U_FACTOR');
v = get(hObject,'Value');
switch v
    case 1
```

```
    U_FACTOR = 0.0254; %Pixeles a centimetros
    set(handles.sizeBroca,'visible', 'on');
    case 2
    U_FACTOR = 0.010; %Pixeles a pulgadas
    set(handles.sizeBroca,'visible', 'on');
    case 3
    U_FACTOR = 1; %Pixeles a pixeles
    set(handles.sizeBroca,'visible', 'off');
end
U_FACTOR_F = U_FACTOR;

if U_FACTOR_0 ~= U_FACTOR_F

    if(length(get(handles.txtRadioFresa,'string'))>0)
        txt                                                                    =
str2num(get(handles.txtRadioFresa,'string'))*U_FACTOR_F/U_FACTOR_0;
        set(handles.txtRadioFresa,'string',txt);
     end
    if(length(get(handles.txtSigma,'string'))>0 )
        txt = str2num(get(handles.txtSigma,'string'))*U_FACTOR_F/U_FACTOR_0;
        set(handles.txtSigma,'string',txt);
    end
    if(length(get(handles.txtG,'string'))>0 )
        txt = str2num(get(handles.txtG,'string'))*U_FACTOR_F/U_FACTOR_0;
        set(handles.txtG,'string',txt);
    end
    if(length(get(handles.txtProfundidad,'string'))>0 )
        txt                                                                    =
str2num(get(handles.txtProfundidad,'string'))*U_FACTOR_F/U_FACTOR_0;
        set(handles.txtProfundidad,'string',txt);
```

```
    end
end
set(handles.text9,'string',    sprintf('%3.1f    <    r    <    %3.1f',6*U_FACTOR,
101*U_FACTOR));
assignin('base','U_FACTOR',U_FACTOR);


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if              ispc              &&              isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function txtRadioFresa_Callback(hObject, eventdata, handles)
% hObject    handle to txtRadioFresa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtRadioFresa as text
%       str2double(get(hObject,'String')) returns contents of txtRadioFresa as a double
r =  str2double(get(handles.txtRadioFresa,'string'));
sigma = (2-1.414)*r;
set(handles.text10,'string',sprintf('> %3.3f',sigma));
```

```
% --- Executes during object creation, after setting all properties.
function txtRadioFresa_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtRadioFresa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function txtSigma_Callback(hObject, eventdata, handles)
% hObject    handle to txtSigma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtSigma as text
%        str2double(get(hObject,'String')) returns contents of txtSigma as a double
r =  str2double(get(handles.txtRadioFresa,'string'));
sigma = str2double(get(handles.txtSigma,'string'));
g = 1.414*r;
set(handles.text11,'string',sprintf('< %3.3f',g));

% --- Executes during object creation, after setting all properties.
function txtSigma_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtSigma (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function txtG_Callback(hObject, eventdata, handles)
% hObject    handle to txtG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtG as text
%        str2double(get(hObject,'String')) returns contents of txtG as a double

% --- Executes during object creation, after setting all properties.
function txtG_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% [fileImagen, pathImagen] = uigetfile('*.bmp;*.jpg;*.dxf','Seleccionar Imagen de la
Cavidad');
% imagen = double(imread(fullfile(pathImagen, fileImagen),'bmp'));
[fileImagen, pathImagen] = uigetfile(...
   {'*.bmp;*.jpg;*.dxf', '*.bmp;*.jpg;*.dxf Todos los archivos de imagen soportados';
...
   '*.bmp','*.bmp Archivo de mapa de bits'; ...
   '*.jpg','*.jpg Archivo JPEG'; ...
   '*.dxf','*.dxf CAD Drawing Exchange File'}, ...
   'Seleccionar Imagen de la Cavidad');
[pathstr, name, ext] = fileparts( [pathImagen fileImagen] );
if( strcmpi(ext,'.dxf') )
   arg = [ '.\Acme\AcmeCADConverter.exe /r /e /ls /p 1 /f 1 ' "" fullfile(pathImagen,
fileImagen) "" ];
   convertir = dos(arg);
   if convertir==0
      fileImagen = [name '.bmp'];
```

```
        im = double(imread(fullfile(pathImagen, fileImagen),'bmp'));
        im = im(:,:,1)>0;
        imt = ones(size(im)+20);
        imt(10:size(im,1)+9,10:size(im,2)+9)=im;
        im = imt;
        borde = edge(im,'canny');
        [X,Y] = find(borde==1);
        minx = min(X);     miny = min(Y);
        maxx = max(X);     maxy = max(Y);
        im = im(minx:maxx,miny:maxy);
        im = im*255;
        imagen(:,:,1) = im;
        imagen(:,:,2) = im;
        imagen(:,:,3) = im;
        imwrite(imagen,'i_Model.bmp','bmp');
        fileImagen = 'i_Model.bmp';
        imagen = double(imread(fullfile(pathImagen, fileImagen),'bmp'));
    end
else
    if( strcmpi(ext,'.bmp') )
        imagen = double(imread(fullfile(pathImagen, fileImagen),'bmp'));
    else
        imagen = double(imread(fullfile(pathImagen, fileImagen),'jpg'));
    end
end

assignin('base','imagen',imagen);
axes(handles.axes4);
imshow(imagen);
set(handles.txtRes,'string',sprintf('Resolución = [ %d x %d ] pixeles',...
```

```
    size(imagen,1),size(imagen,2)));

if (size(imagen,1)>500 || size(imagen,2)> 500)
    assignin('base','isHighRes', 1);
    imtool(imagen);
else
    assignin('base','isHighRes', 0);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4
cla;

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes2


% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3

% --- Executes during object creation, after setting all properties.
function txtRes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtRes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

function txtProfundidad_Callback(hObject, eventdata, handles)
% hObject    handle to txtProfundidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtProfundidad as text
%        str2double(get(hObject,'String')) returns contents of txtProfundidad as a
double

% --- Executes during object creation, after setting all properties.
function txtProfundidad_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to txtProfundidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selection change in sizeBroca.
function sizeBroca_Callback(hObject, eventdata, handles)
% hObject    handle to sizeBroca (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns sizeBroca contents as cell array
%        contents{get(hObject,'Value')} returns selected item from sizeBroca
v = get(hObject,'Value');
w = get(handles.popupmenu1,'Value');
if w==1
    U_FACTOR = 2.540;
```

```
else
   U_FACTOR = 1;
end
switch v
   case 1 %1/8
   set(handles.txtRadioFresa,'string',0.125*U_FACTOR);
   case 2 %1/4
   set(handles.txtRadioFresa,'string', 0.25*U_FACTOR);
   case 3 %3/8
   set(handles.txtRadioFresa,'string', (3/8)*U_FACTOR);
   case 4 %1/2
   set(handles.txtRadioFresa,'string', 0.5*U_FACTOR);
   case 5 %5/8
   set(handles.txtRadioFresa,'string',(5/8)*U_FACTOR);
   case 6 %3/4
   set(handles.txtRadioFresa,'string', 0.75*U_FACTOR);
   case 7 %7/8
   set(handles.txtRadioFresa,'string', (7/8)*U_FACTOR);
   case 8 % 1
   set(handles.txtRadioFresa,'string', 1*U_FACTOR);
   case 9 %1-1/4
   set(handles.txtRadioFresa,'string',1.25*U_FACTOR);
   case 10 %1-1/2
   set(handles.txtRadioFresa,'string', 1.5*U_FACTOR);
   case 11 %1-3/4
   set(handles.txtRadioFresa,'string', 1.75*U_FACTOR);
   case 12 % 2
   set(handles.txtRadioFresa,'string', 2*U_FACTOR);
end
set(handles.porcentaje,'string','');
```

```
set(handles.txtSigma,'string','');
set(handles.txtG,'string','');


% --- Executes during object creation, after setting all properties.
function sizeBroca_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sizeBroca (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function porcentaje_Callback(hObject, eventdata, handles)
% hObject    handle to porcentaje (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of porcentaje as text
%        str2double(get(hObject,'String')) returns contents of porcentaje as a double
r =  str2double(get(handles.txtRadioFresa,'string'));
porcentaje = str2double(get(handles.porcentaje,'string'));
set(handles.txtSigma,'string',r*porcentaje/100);
set(handles.txtG,'string',r-(r*porcentaje/100));

% --- Executes during object creation, after setting all properties.
```

```
function porcentaje_CreateFcn(hObject, eventdata, handles)
% hObject    handle to porcentaje (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --- Executes when figure1 is resized.
function figure1_SizeChangedFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

Codigos step 2

```
function varargout = Step2(varargin)
% STEP2 MATLAB code for Step2.fig
%      STEP2, by itself, creates a new STEP2 or raises the existing
%      singleton*.
%
%      H = STEP2 returns the handle to a new STEP2 or the handle to
%      the existing singleton*.
%
%      STEP2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STEP2.M with the given input arguments.
%
%      STEP2('Property','Value',...) creates a new STEP2 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Step2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Step2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help Step2

% Last Modified by GUIDE v2.5 16-Oct-2017 18:56:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Step2_OpeningFcn, ...
                   'gui_OutputFcn',  @Step2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Step2 is made visible.
function Step2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to Step2 (see VARARGIN)

% Choose default command line output for Step2
handles.output = hObject;
scrsz = get(0,'ScreenSize');
pos_act = get(gcf, 'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

% Update handles structure

axes(handles.axes1);
img = imread('imagenes/red_neural.jpg');
imshow(img);
axes(handles.axes3);
fig = imread('imagenes/next.jpg');
imshow(fig);
axes(handles.axes4); cla;
if(evalin('base','isHighRes')==0)
   imshow(evalin('base', 'imagen'));
else
   set(handles.axes4,'visible','off');
end
%============================
set(handles.figure1,'resize','on');
%============================
handles.output = hObject;
```

```
guidata(hObject, handles);
% UIWAIT makes Step2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Step2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbuttonBack.
function pushbuttonBack_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonBack (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h2');
mostrarVentana('h1');

% --- Executes on button press in pushbuttonNext.
function pushbuttonNext_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonNext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h2');
```

```
h3 = Step3;
assignin('base','h3',h3);

% --- Executes on button press in pushbuttonExit.
function pushbuttonExit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
choice = questdlg('¿Realmente desea salir?','Salir','Si','No','No');
switch choice
   case 'Si'
   clear all;
   clc;
   close;
end

% --- Executes on button press in pushbuttonHome.
function pushbuttonHome_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonHome (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h2');
h1 = Step1;

% --- Executes during object creation, after setting all properties.
function pushbuttonExit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% image_rgb = imread('imagenes/icon.jpg');
```

```
% set(hObject,'cdata',image_rgb);


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4

% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3

% --- Executes during object creation, after setting all properties.
function txtRes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtRes (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in pushbuttonIdentificar.
function pushbuttonIdentificar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonIdentificar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
radioFresa = evalin('base','radioFresa');
g = evalin('base','g');
imagen = evalin('base','imagen');
axes(handles.axes4); cla;
isHighRes = evalin('base', 'isHighRes');
posx=str2double(get(handles.edit5,'string'));%mover el inicio de los puntos
posy=str2double(get(handles.edit4,'string'));
assignin('base','posx', posx);
assignin('base','posy', posy);

if isHighRes == 1
   imtool(imagen);
   generarRejillaBig(radioFresa,g,imagen, imgca);
else
   generarRejilla(radioFresa,g,imagen);
end

U_FACTOR = evalin('base','U_FACTOR');
p = evalin('base','p')*U_FACTOR;
set(handles.uitable1,'data',p);
```

```
% --- Executes during object creation, after setting all properties.
function uitable1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

Codigos del step 3.

```
function varargout = Step3(varargin)
% STEP3 MATLAB code for Step3.fig
%      STEP3, by itself, creates a new STEP3 or raises the existing
%      singleton*.
%
%      H = STEP3 returns the handle to a new STEP3 or the handle to
%      the existing singleton*.
%
%      STEP3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STEP3.M with the given input arguments.
```

```
%      STEP3('Property','Value',...) creates a new STEP3 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Step3_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Step3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Step3

% Last Modified by GUIDE v2.5 10-Oct-2017 16:46:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Step3_OpeningFcn, ...
                   'gui_OutputFcn',  @Step3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

 % --- Executes just before Step3 is made visible.
function Step3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Step3 (see VARARGIN)

% Choose default command line output for Step3
handles.output = hObject;
scrsz = get(0,'ScreenSize');
pos_act = get(gcf, 'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

% Update handles structure

axes(handles.axes1);
img = imread('imagenes/red_neural.jpg');
imshow(img);

axes(handles.axes3);
```

```
fig = imread('imagenes/next.jpg');
imshow(fig);

imtool close all;
axes(handles.axes4); cla;
set(handles.axes4, 'visible','off');
imagen = evalin('base','imagen');
radioFresa = evalin('base', 'radioFresa');
fig = evalin('base','rejilla');
isHighRes = evalin('base','isHighRes');
if isHighRes == 0
    cla;
    set(handles.axes4, 'visible','on');
    imshow(imagen(:,:,1)); hold on;
    plot(fig(:,2),fig(:,1),'r.');
    plot(fig(:,2),fig(:,1),'go','MarkerSize',12*fix(2*radioFresa)/16);
else
    imtool(imagen);
    assignin('base', 'hIma', imgca);
    hold(imgca,'on');
    plot(imgca, fig(:,2),fig(:,1),'go','MarkerSize',12*fix(2*radioFresa)/16);
    plot(imgca, fig(:,2),fig(:,1),'r.');
end
%============================
set(handles.figure1,'resize','on');
%============================
handles.output = hObject;
guidata(hObject, handles);
% UIWAIT makes Step3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Step3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbuttonBack.
function pushbuttonBack_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonBack (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h3');
mostrarVentana('h2');

% --- Executes on button press in pushbuttonNext.
function pushbuttonNext_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonNext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h3');
h4 = Step4;
assignin('base','h4',h4);

 % --- Executes on button press in pushbuttonExit.
function pushbuttonExit_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
choice = questdlg('¿Realmente desea salir?','Salir','Si','No','No');
switch choice
    case 'Si'
    clear all;
    clc;
    close;
end



% --- Executes on button press in pushbuttonHome.
function pushbuttonHome_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonHome (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h3');
h1 = Step1;

% --- Executes during object creation, after setting all properties.
function pushbuttonExit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% image_rgb = imread('imagenes/icon.jpg');
% set(hObject,'cdata',image_rgb);
```

```
% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4


% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3


% --- Executes during object creation, after setting all properties.
function txtRes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtRes (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in pushbuttonIdentificar.
function pushbuttonIdentificar_Callback(hObject, eventdata, handles)% esta es la de cohonen
% hObject    handle to pushbuttonIdentificar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p = evalin('base','p');
w = evalin('base','w');
U_FACTOR = evalin('base','U_FACTOR');
imagenOriginal = evalin('base','imagen');
radioFresa = evalin('base','radioFresa');
imtool close all;
axes(handles.axes4);
hA = gca;
if (evalin('base', 'isHighRes')==0)
  [ruta, indices] = solveTSP(w, true, gca, imagenOriginal, radioFresa, 1);
  hI = hA;
else
  imtool(imagenOriginal);
  hI = imgca;
  [ruta, indices] = solveTSPBig(w, true, hI, hA, imagenOriginal, radioFresa, 1);
end
assignin('base','ruta', ruta);
assignin('base','indices', indices);
set(handles.uitable1,'data',ruta*U_FACTOR);
set(handles.uitable1,'Enable','on');
```

```
set(handles.pushbuttonIdentificar,'Enable','on');
mask = correccionTrayectoriaSOM(ruta,imagenOriginal,size(ruta,1),hI);
assignin('base','mask', mask);
timetot = evalin('base', 'timetot');
iteracion = evalin('base', 'iteracion');
distanciatot = evalin('base', 'distanciatot');
set(handles.text21,'string',distanciatot);
set(handles.text23,'string',iteracion);
set(handles.text25,'string',timetot(1,4:6));


% --- Executes on button press in pushbuttonEntrenar.
function pushbuttonEntrenar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonEntrenar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes4);
hA = gca;
axis off;
p = evalin('base','p');
U_FACTOR = evalin('base', 'U_FACTOR');
radioFresa = evalin('base','radioFresa');
imagenOriginal = evalin('base','imagen');
set(handles.uitable1,'data',p*U_FACTOR);
set(handles.uitable1,'Enable','on');
if evalin('base','isHighRes') == 1
   w                                                                    =
optimizarTrayectoriaSOMBig(p,radioFresa,(imagenOriginal(:,:,1)==255),evalin('bas
e','hIma'), hA);
else
```

```
    w = optimizarTrayectoriaSOM(p,radioFresa,(imagenOriginal(:,:,1)==255));
end
assignin('base','w',w);
set(handles.uitable1,'data',w*U_FACTOR);
set(handles.uitable1,'Enable','on');
%set(handles.pushbuttonIdentificar,'Enable','on');
%////////////////////////////////////7}
p = evalin('base','p');
w = evalin('base','w');
U_FACTOR = evalin('base','U_FACTOR');
imagenOriginal = evalin('base','imagen');
radioFresa = evalin('base','radioFresa');
imtool close all;
axes(handles.axes4);
hA = gca;
if (evalin('base', 'isHighRes')==0)
  [ruta, indices] = solveTSP(w, true, gca, imagenOriginal, radioFresa, 1);
  hI = hA;
else
  imtool(imagenOriginal);
  hI = imgca;
  [ruta, indices] = solveTSPBig(w, true, hI, hA, imagenOriginal, radioFresa, 1);
end
assignin('base','ruta', ruta);
assignin('base','indices', indices);
set(handles.uitable1,'data',ruta*U_FACTOR);
set(handles.uitable1,'Enable','on');
%set(handles.pushbuttonIdentificar,'Enable','on');
mask = correccionTrayectoriaSOM(ruta,imagenOriginal,size(ruta,1),hI);
assignin('base','mask', mask);
```

```
timetot = evalin('base', 'timetot');
iteracion1 = evalin('base', 'iteracion1');
iteracion2 = evalin('base', 'iteracion2');
ittot=iteracion1+iteracion2;
distanciatot = evalin('base', 'distanciatot');
set(handles.text21,'string',distanciatot);
set(handles.text23,'string',ittot);
set(handles.text25,'string',timetot(1,4:6));



% --- Executes during object creation, after setting all properties.
function uitable1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)% aca ya es hopfield7777//////////////////
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes4);
hA = gca;
axis off;
p = evalin('base','p');
U_FACTOR = evalin('base', 'U_FACTOR');
radioFresa = evalin('base','radioFresa');
```

```
imagenOriginal = evalin('base','imagen');
set(handles.uitable1,'data',p*U_FACTOR);
set(handles.uitable1,'Enable','on');
if evalin('base','isHighRes') == 1
   w2 = tsphopf_big(p,radioFresa,U_FACTOR,evalin('base','hIma'),
hA,(imagenOriginal(:,:,1)==255));
   hI = imgca;
else
   w2 = tsphopf(p,radioFresa,U_FACTOR);
   hI = hA;
end
assignin('base','w2',w2);
set(handles.uitable1,'data',w2*U_FACTOR);
set(handles.uitable1,'Enable','on');
%set(handles.pushbuttonIdentificar,'Enable','on');
assignin('base','ruta',w2*U_FACTOR);
%timetot = evalin('base', 'timetot');
%iteracion = evalin('base', 'iteracion');
distanciatot = evalin('base', 'distanciatot');
set(handles.text21,'string',distanciatot);
%set(handles.text23,'string',iteracion);
%set(handles.text25,'string',timetot(1,4:6));
%//////////////////////////////////////////////////78+
w2 = evalin('base','w2');
U_FACTOR = evalin('base','U_FACTOR');
imagenOriginal = evalin('base','imagen');
radioFresa = evalin('base','radioFresa');
imtool close all;
axes(handles.axes4);
hA = gca;
```

```
if (evalin('base', 'isHighRes')==0)
  [ruta, indices] = solveTSP(w2, true, gca, imagenOriginal, radioFresa, 1);
  hI = hA;
else
  imtool(imagenOriginal);
  hI = imgca;
  [ruta, indices] = solveTSPBig(w2, true, hI, hA, imagenOriginal, radioFresa, 1);
end
assignin('base','ruta', ruta);
assignin('base','indices', indices);
set(handles.uitable1,'data',ruta*U_FACTOR);
set(handles.uitable1,'Enable','on');
%set(handles.pushbuttonIdentificar,'Enable','on');
mask = correccionTrayectoriaSOM(ruta,imagenOriginal,size(ruta,1),hI);
assignin('base','mask', mask);
timetot = evalin('base', 'timetot');
iteracion1 = evalin('base', 'iteracion1');
iteracion2 = evalin('base', 'iteracion2');
ittot=iteracion1+iteracion2;
distanciatot = evalin('base', 'distanciatot');
set(handles.text21,'string',distanciatot);
set(handles.text23,'string',ittot);
set(handles.text25,'string',timetot(1,4:6));

% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
w2 = evalin('base','w2');
U_FACTOR = evalin('base','U_FACTOR');
imagenOriginal = evalin('base','imagen');
radioFresa = evalin('base','radioFresa');
imtool close all;
axes(handles.axes4);
hA = gca;
if (evalin('base', 'isHighRes')==0)
  [ruta, indices] = solveTSP(w2, true, gca, imagenOriginal, radioFresa, 1);
  hI = hA;
else
  imtool(imagenOriginal);
  hI = imgca;
  [ruta, indices] = solveTSPBig(w2, true, hI, hA, imagenOriginal, radioFresa, 1);
end
assignin('base','ruta', ruta);
assignin('base','indices', indices);
set(handles.uitable1,'data',ruta*U_FACTOR);
set(handles.uitable1,'Enable','on');
set(handles.pushbuttonIdentificar,'Enable','on');
mask = correccionTrayectoriaSOM(ruta,imagenOriginal,size(ruta,1),hI);
assignin('base','mask', mask);
timetot = evalin('base', 'timetot');
iteracion = evalin('base', 'iteracion');
```

```
distanciatot = evalin('base', 'distanciatot');
set(handles.text21,'string',distanciatot);
set(handles.text23,'string',iteracion);
set(handles.text25,'string',timetot(1,4:6));


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes4);
hA = gca;
axis off;
p = evalin('base','p');
U_FACTOR = evalin('base', 'U_FACTOR');
radioFresa = evalin('base','radioFresa');
imagenOriginal = evalin('base','imagen');
set(handles.uitable1,'data',p*U_FACTOR);
set(handles.uitable1,'Enable','on');
if evalin('base','isHighRes') == 1
   w = tsp_programing(p,U_FACTOR);
   hI = imgca;
else
   w = tsp_programing(p,U_FACTOR);
   hI=hA;
end
assignin('base','w',w);
set(handles.uitable1,'data',w*U_FACTOR);
set(handles.uitable1,'Enable','on');
```

```
%set(handles.pushbuttonIdentificar,'Enable','on');
ruta=evalin('base', 'ruta');
assignin('base','ruta',ruta);
mask = correccionTrayectoriaSOM(ruta,imagenOriginal,size(ruta,1),hI);%esto es por solo provar
assignin('base','mask',mask);
timetot = evalin('base', 'timetot');
iteracion = evalin('base', 'iteracion');
distanciatot = evalin('base', 'distanciatot');
set(handles.text21,'string',distanciatot);
set(handles.text23,'string',iteracion);
set(handles.text25,'string',timetot(1,4:6));



% --------------------------------------------------------------------
function uipanel1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Codigos Step 4.

```
function varargout = Step4(varargin)
% STEP4 MATLAB code for Step4.fig
%      STEP4, by itself, creates a new STEP4 or raises the existing
%      singleton*.
%
%      H = STEP4 returns the handle to a new STEP4 or the handle to
%      the existing singleton*.
%
```

```
%      STEP4('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STEP4.M with the given input arguments.
%
%      STEP4('Property','Value',...) creates a new STEP4 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Step4_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Step4_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Step4

% Last Modified by GUIDE v2.5 22-Oct-2017 17:50:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Step4_OpeningFcn, ...
                   'gui_OutputFcn',  @Step4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Step4 is made visible.
function Step4_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Step4 (see VARARGIN)

% Choose default command line output for Step4
handles.output = hObject;
scrsz = get(0,'ScreenSize');
pos_act = get(gcf, 'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

% Update handles structure

axes(handles.axes1);
img = imread('imagenes/red_neural.jpg');
```

```
imshow(img);

imtool close all;
axes(handles.axes4); cla;
set(handles.axes4, 'visible', 'off');
if(evalin('base','isHighRes')==0)
    fig = evalin('base','imagen');
    imshow(fig(:,:,1)); hold on;
    radioFresa = evalin('base', 'radioFresa');
    fig = evalin('base','rejilla');
    plot(fig(:,2),fig(:,1),'r.');
    plot(fig(:,2),fig(:,1),'go','MarkerSize',12*fix(2*radioFresa)/16);
end

axes(handles.axes3);
fig = imread('imagenes/next.jpg');
imshow(fig);
%============================
set(handles.figure1,'resize','on');
%============================
handles.output = hObject;
guidata(hObject, handles);
% UIWAIT makes Step4 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Step4_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in pushbuttonBack.
function pushbuttonBack_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonBack (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h4');
mostrarVentana('h3');

% --- Executes on button press in pushbuttonNext.
function pushbuttonNext_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonNext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% assignin('base','radioFresa',str2double(get(handles.txtRadioFresa,'string')));
% assignin('base','sigma',str2double(get(handles.txtSigma,'string')));
% assignin('base','g',str2double(get(handles.txtG,'string')));
% h3 = Step3;
% assignin('base','h3',h3);
figure;
grid on;
ruta = evalin('base', 'ruta');
indice = evalin('base', 'indices');
radioFresa = evalin('base', 'radioFresa');
```

```
imagen = evalin('base','imagen');
prof = evalin('base','profundidad');
mask = evalin('base', 'mask');
profcort= str2double(get(handles.edit4,'string'));

testGraficar(imagen, ruta, indice, mask, radioFresa,prof,profcort);
fid = fopen('puntos.txt', 'w');
fprintf(fid, '%.2f \t %.2f \n\r\n', ruta);
fclose(fid);dos('start puntos.txt');



% --- Executes on button press in pushbuttonExit.
function pushbuttonExit_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
choice = questdlg('¿Realmente desea salir?','Salir','Si','No','No');
switch choice
    case 'Si'
    clear all;
    clc;
    close;
end



% --- Executes on button press in pushbuttonHome.
function pushbuttonHome_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonHome (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ocultarVentana('h4');
h1 = Step1;

% --- Executes during object creation, after setting all properties.
function pushbuttonExit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbuttonExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% image_rgb = imread('imagenes/icon.jpg');
% set(hObject,'cdata',image_rgb);


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4
```

```
% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3


% --- Executes during object creation, after setting all properties.
function txtRes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtRes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in pushbuttonIdentificar.
function pushbuttonIdentificar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonIdentificar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes4);
if (evalin('base', 'isHighRes') ==0)
   hI = gca;
else
   imtool(evalin('base', 'imagen'));
   hI = imgca;
   hold(hI,'on');
```

```
    radioFresa = evalin('base', 'radioFresa');
    fig = evalin('base','rejilla');
    plot(hI,fig(:,2),fig(:,1),'r.');
    plot(hI,fig(:,2),fig(:,1),'go','MarkerSize',12*fix(2*radioFresa)/16);
end
ruta = evalin('base','ruta');
indices = evalin('base','indices');
radioFresa = evalin('base','radioFresa');
mask = evalin('base','mask');
hold(hI,'on');
plot(hI,ruta(1,2), ruta(1,1), 'r.', 'MarkerSize',20);
for i=1:size(indices,1)-1
    if(mask(i)==1)
    hold(hI,'on');
    P1 = [ ruta(i,1) ruta(i+1,1) ];
    P2 = [ ruta(i,2) ruta(i+1,2) ];
    plot(hI, P2,P1,'b-', 'LineWidth',12*fix(radioFresa)/16);
    if(rem(i,4)==0)
        pause(0.1);
    end
    end
end
plot(hI,ruta(1,2), ruta(1,1), 'r.', 'MarkerSize',25);
plot(hI,ruta(end,2), ruta(end,1), 'b.', 'MarkerSize',25);
for i=1:size(indices,1)-1
    if(mask(i)==1)
        hold(hI,'on');
        P1 = [ ruta(i,1) ruta(i+1,1) ];
        P2 = [ ruta(i,2) ruta(i+1,2) ];
        plot(hI,P2,P1,'k-');
```

```
    end
end


% --- Executes on button press in pushbuttonEntrenar.
function pushbuttonEntrenar_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonEntrenar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p = evalin('base','p');
imagenOriginal = evalin('base','imagen');
radioFresa = evalin('base','radioFresa');
axes(handles.axes4);
if (evalin('base','isHighRes')==0)
    hI = gca;
else
    imtool(evalin('base', 'imagen'));
    hI = imgca;
    hold(hI,'on');
    radioFresa = evalin('base', 'radioFresa');
    fig = evalin('base','rejilla');
    plot(hI,fig(:,2),fig(:,1),'r.');
    plot(hI,fig(:,2),fig(:,1),'go','MarkerSize',12*fix(2*radioFresa)/16);
end
ruta= evalin('base','ruta');
mask= evalin('base','mask');
indices = evalin('base','indices');
dreal=0;
for i=1:size(indices,1)-1
    if(mask(i)==1)
```

```
    hold(hI,'on');
    P1 = [ ruta(i,1) ruta(i+1,1) ];
    P2 = [ ruta(i,2) ruta(i+1,2) ];
    plot(hI, P2,P1,'k-');
    dreal=dreal+sqrt((ruta(i,1)-ruta(i+1,1))^2+(ruta(i,2)-ruta(i+1,2))^2);
    end
end
U_FACTOR = evalin('base','U_FACTOR');
distancia = evalin('base','distanciatot');
set(handles.uitable1,'data',ruta*U_FACTOR);
set(handles.uitable1,'Enable','on');
set(handles.pushbuttonIdentificar,'Enable','on');
assignin('base','dreal', dreal*U_FACTOR);
set(handles.text21,'string',dreal*U_FACTOR);
efecti=(dreal*U_FACTOR)/distancia*100;
set(handles.text23,'string',efecti);



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
avance = str2double(get(handles.edit5,'string'));
longitud = str2double(get(handles.text21,'string'));
revoluciones = str2double(get(handles.edit6,'string'));
prof_corte = str2double(get(handles.edit4,'string'));
prof = evalin('base','profundidad');

%tiempo de mecanizado
tiempo_mec=(longitud)/(avance*revoluciones);
%# de cortes
num_cor=ceil(prof/prof_corte);
%tiempo total
tiempo_total=tiempo_mec*num_cor;

set(handles.text27,'string',tiempo_mec);
set(handles.text29,'string',num_cor);
set(handles.text28,'string',tiempo_total);
```

Anexo F. COMPARACION DE CANTIDAD DE ITERACIONES

Una comparación bastante contundente de lo que representa la complejidad del problema del agente viajero se hace en la tabla 16, tomando como base el computador convencional el cual realiza un promedio de 27100 millones de operaciones por segundo y el supercomputador más potente del mundo el cual realiza 93 cuatrillones de operaciones por segundo

Tabla 16. Tiempos de solución Agente Viajero

Comparacion de tiempos de solucion					
# de puntos	posibles rutas o caminos	Tiempo en años (computador convencional)	Tiempo en min (computador convencional)	Tiempo en años (supercomputador)	Tiempo en min (supercomputador)
3	1,0E+00	1,1701E-18	6,15006E-13	3,40966E-34	1,79211E-28
5	1,2E+01	1,40412E-17	7,38007E-12	3,80518E-31	2E-25
7	3,6E+02	4,21237E-16	2,21402E-10	1,14155E-29	6E-24
9	2,0E+04	2,35893E-14	1,23985E-08	6,39269E-28	3,36E-22
11	1,8E+06	2,12303E-12	1,11587E-06	5,75342E-26	3,024E-20
13	2,4E+08	2,80241E-10	0,000147294	7,59452E-24	3,99168E-18
15	4,4E+10	5,10038E-08	0,026807593	1,3822E-21	7,26486E-16
17	1,0E+13	1,22409E-05	6,433822229	3,31729E-19	1,74357E-13
19	3,2E+15	0,003745718	1968,749602	1,01509E-16	5,33531E-11
21	1,2E+18	1,423373	748124,8488	3,85734E-14	2,02742E-08
23	5,6E+20	657,598326	345633680,1	1,78209E-11	9,36667E-06
25	3,1E+23	362994,2759	1,9079E+11	9,83714E-09	0,005170403
27	2,0E+26	235946279,4	1,24013E+14	6,39414E-06	3,360762176
29	1,5E+29	1,78375E+11	9,37541E+16	0,004833973	2540,736205
31	1,3E+32	1,55187E+14	8,15661E+19	4,205556504	2210440,498
50	3,0E+62	3,55876E+44	1,87049E+50	9,64425E+30	5,06902E+36
100	4,7E+155	5,4601E+137	2,8698E+143	1,4797E+124	7,7772E+129
150	1,9E+260	2,2284E+242	1,1713E+248	6,039E+228	3,1741E+234
200	#¡NUM!	#¡NUM!	#¡NUM!	#¡NUM!	#¡NUM!

En la figura 119 se puede observar un error presentado por el ordenador debido a que la memoria de procesamiento no fue suficiente para ejecutar correctamente la cantidad de datos solicitada.

Figura 119. Error de incapacidad de procesamiento