INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN Y DESARROLLO
DE TECNOLOGÍA DIGITAL

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

"OPTIMIZACIÓN DE TRAYECTORIAS PARA MÁQUINAS DE CONTROL NUMÉRICO MEDIANTE COLONIA DE HORMIGAS"

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS EN SISTEMAS DIGITALES

P R E S E N T A:

ING. NATALY MEDINA RODRÍGUEZ

BAJO LA DIRECCIÓN DE:

DR. OSCAR H. MONTIEL ROSS **DR. ROBERTO SEPÚLVEDA CRUZ**

JUNIO DE 2011 TIJUANA, B.C., MÉXICO

SIP-14-BIS

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de Tijuana, B.C. siendo las 11:00 horas del día 16 del mes de junio del 2011 se reunieron los miembros de la Comisión Revisora de Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de CITEDI para examinar la tesis titulada:

OPTIMIZACIÓN DE TRAYECTORIAS PARA MAQUINAS DE CONTROL NUMERICO (CNC) MEDIANTE COLONIAS DE HORMIGAS.

Presentada por el alumno:

MEDINA	**RODRIGUEZ**	**NATALY**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	0	9	1	8	4	2

aspirante de:

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

Después de intercambiar opiniones, los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de tesis

DR. OSCAR HUMBERTO MONTIEL ROSS

DR. ROBERTO SEPÚLVEDA CRUZ

DR. JUAN JOSÉ TAPIA ARMENTA

DR. KONSTANTIN STARKOV

DR. MOISÉS SÁNCHEZ ADAME

PRESIDENTE DEL COLEGIO DE PROFESORES

DR. LUIS ARTURO GONZÁLEZ HERNÁNDEZ

S. E. P.
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION Y DESARROLLO
DE TECNOLOGIA DIGITAL
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de Tijuana, B.C el día 20 del mes junio del año 2011, el (la) que suscribe Nataly Medina Rodríguez alumno (a) del Programa de Maestría en Ciencias en Sistemas Digitales con número de registro B091842, adscrito al Centro de Investigación y Desarrollo de Tecnología Digital, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Oscar H. Montiel Ross y Dr. Roberto Sepúlveda Cruz y cede los derechos del trabajo intitulado Optimización de Trayectorias para Máquinas de Control Numérico mediante Colonia de Hormigas, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección Av. Del parque No. 1310, mesa de Otay, Tijuana, B.C, C.P. 22510 México. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Nataly Medina Rodríguez

Nombre y firma

"Just because something doesn't do what you planned it to do doesn't mean it's useless."

- Thomas Alva Edison

El siguiente trabajo de tesis se encuentra

dedicado a Dios primeramente,

y a todas aquellas personas que forman

parte importante en mi vida,

sobre todo a mis padres,

y a mi compañero.

Agradecimientos

Agradezco al Instituto Politécnico Nacional, al Centro de Investigación y Desarrollo de Tecnología Digital y al Consejo Nacional de Ciencia y Tecnología por haberme permitido formar parte del equipo de investigación para el desarrollo de la presente tesis. Al Dr. Oscar H. Montiel Ross por haberme apoyado y orientado en mi trabajo de tesis así como al Dr. Roberto Sepúlveda Cruz; a todo el comité de titulación conformado por el Dr. Moisés Sánchez Adame, Dr. Juan Tapia Armenta, Dr. Konstantin Starkov, por brindarme su apoyo durante la investigación. A mi compañero Francisco J. Díaz Delgadillo, por apoyarme durante mi investigación, estar al pendiente del progreso del presente trabajo. Agradezco a CETYS Universidad, campus Tijuana por haber formado parte de mi investigación, al permitir el uso de una máquina CNC Haas® Automation con la que la institución cuenta y toda clase de asesoría para el manejo de la máquina otorgada por Gerardo Ramírez. Al maestro Roberto Salas, Jaime Ramos, catedráticos de esta misma institución educativa por haberme otorgado su apoyo para concluir con este proceso. Finalmente, agradezco especialmente a mis padres, Juan Medina y Rosa Ma. Rodríguez, por todo su apoyo que como siempre me han brindado, por su cariño y afecto que me permitieron concluir este nuevo logro en mi vida.

Contenido

Lista de Figuras

Lista de Tablas

Lista de acrónimos

Acrónimo	Significado
ACO	Ant Colony Optimization
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CNC	Computer Numerical Control
CTTP	Cutting Tool Travel Path
DXF	Drawing Interchange Format
NC	Numeric Control
PCB	Printed Circuit Board
PCS	Pixel Coordinate System
TSP	Travelling Salesman Problem
WCS	World Coordinate System

OPTIMIZACIÓN DE TRAYECTORIAS PARA MÁQUINAS DE CONTROL NUMÉRICO MEDIANTE COLONIA DE HORMIGAS

Resumen

Una máquina CNC es una herramienta o dispositivo que es de suma importancia para un sistema industrial; es utilizado en el diseño y manufactura de un producto. La productividad de las herramientas de máquinas CNC han mejorado utilizando sistemas basados en CAD/CAM para la generación de programas NC. Actualmente, diversos paquetes comerciales CAD/CAM que generan programas NC han sido desarrollados y aplicados a diversos procesos de corte. Uno de los procesos de corte mediante CNC es el proceso de taladrado.

En este trabajo se pretende encontrar una solución eficiente para determinar la mejor secuencia de operaciones para un conjunto de orificios que se encuentran ubicados de manera asimétrica en una placa PCB. Para encontrar esta secuencia de operaciones de taladrado que presentan la ruta más corta para la herramienta de corte, se ha utilizado la optimización por Colonia de Hormigas. Después de que la ruta de la herramienta es optimizada, se genera el programa numérico con código G. Esta aplicación puede ser formulada como un caso especial del problema del Agente Viajero (TSP).

Palabras clave: *Optimización por Colonia de Hormigas, ACO, Control Numérico por Computadora, CNC, Problema del Agente Viajero, TSP, taladrado,*

TOOLPATH OPTIMIZATION FOR COMPUTER NUMERICAL CONTROL MACHINES BASED ON ANT COLONY

Abstract

CNC machining is a tool or a device that is critical to an industrial system; it is used to aid the design and the manufacturing of a product. The productivity of CNC machine tools is significantly improved by used CAD/CAM systems for NC program generation. Currently, many CAD/CAM packages that provide automatic NC programming have been developed to various cutting processes. One of the cutting processes machined by CNC machine tools is hole-cutting operations or drilling.

In this work we attempt to find an efficient solution approach to determine the best sequence of G commands for a set of holes in a PCB board. In order to find the hole-cutting sequence that achieves the shortest cutting tool travel path, Ant Colony Optimization is introduced. After the cutting tool travel path is optimized, the generated G-codes are used to code the part of program for this process. This application can be formulated as a special case of the traveling salesman problem (TSP).

Keywords: *Ant Colony Optimization, ACO, Computer Numerical Control, CNC, Traveling Salesman, TSP, drilling,*

Capítulo 1

Introducción

Las líneas de producción son una clase importante en los sistemas de manufactura cuando se van a elaborar grandes cantidades de productos idénticos o similares. Son convenientes para realizar un trabajo en la parte o producto que requiere muchos pasos separados. Entre los ejemplos se encuentran los productos ensamblados, así como las partes maquinadas que se producen en forma masiva, en las cuales se requieren múltiples operaciones de maquinado.

Siendo esta última objeto de estudio en la presente investigación como aplicación industrial, así como la optimización de trayectorias en el proceso de maquinado de piezas que presentan una cantidad de perforaciones por medio de metaheurísticas.

Las perforaciones se han realizado con ayuda de una máquina de control numérico o CNC por sus siglas en inglés, optimizando la ruta que seguiría el eje de perforación y de corte mejor conocido como *spindle*.

Dentro del proceso de maquinado en CNC se deben considerar diversos aspectos tales como el tipo de material y el tipo de herramienta a utilizar con la finalidad de efectuar un mejor proceso de fabricación, además de otros aspectos tales como las

velocidades para el *spindle*, que son objeto de estudio de diversas investigaciones que forman parte del estado del arte de la presente investigación.

El diseño de la plataforma de software tiene la capacidad de generar el código numérico que contiene la información requerida para seguir una trayectoria, siendo el mejor caso de optimización el generar una ruta con la menor distancia posible; esto ha posibilitado realizar el maquinado de un gran número de piezas en el menor tiempo posible.

Cabe destacar que la generación de la trayectoria óptima se realiza mediante la optimización por Colonia de Hormigas, dicho algoritmo ha sido implementado utilizando computación paralela para efectos de reducir tiempos de ejecución y por ende, reducir complejidad computacional.

Existen diversos trabajos relacionados a la paralelización del algoritmo por Colonia de Hormigas, tal es el caso que presentan Ling Chen, Hai-Ying, y Shu Wang [38] con un algoritmo por Colonia de Hormigas adaptativo, denominado como PACO, el cual dividen en grupos P a una serie de hormigas que son ubicadas en P procesadores, el cual no se hace de manera aleatoria la ubicación de las hormigas dentro de los procesadores, para ellos existe un procedimiento adaptativo para hacer uso eficiente de los recursos de hardware.

Para Dongdong, Guanghong y Han Liang [39], presentan una alternativa para la optimización por Colonia de Hormigas mediante cómputo paralelo utilizando OpenMP, comentando que cuentan con una serie de directivas que facilitan al usuario definir

explícitamente la paralelización del algoritmo. Realizan una comparación entre utilizar OpenMP y TBB [39], que es una librería de C++ introducida por Intel®. En el trabajo presentado por [14], muestran la comparación entre diferentes modelos de paralelización, tal es el caso de pase de mensaje o memoria compartida para la optimización por Colonia de Hormigas.

Por otra parte, diversos trabajos de investigación como lo menciona P.W. Prickett y Jian Wang [1] han tenido como objeto de estudio la optimización de parámetros de control de una máquina de control numérico para generar una trayectoria de corte óptima. En este documento se presentan tres consideraciones a tomar para generar una ruta óptima, las cuales se relacionan con las siguientes tres consideraciones:

- **Capacidad de las herramientas.** Según los autores, se dice que la capacidad de las herramientas se encuentra relacionada con el área de trabajo, la ubicación de la herramienta y su descripción, la capacidad de maquinado en general y la capacidad misma de acuerdo a la geometría del diseño.
- **Atributos de los componentes.** Son los atributos físicos de los componentes que serán maquinados, tales como el tipo de material, la superficie de acabado y las tolerancias.
- **Tipo de proceso.** Son las opciones para producir un agujero considerando el método más apropiado y la herramienta asociada.

La investigación realizada por P.W. Prickett y Jian Wang afirma que para una programación óptima de la ruta, se deben considerar dichos aspectos, siendo éstos entradas hacia un sistema experto basado en redes neuronales. La ruta optimizada es

generada para minimizar el número de cambios de herramienta necesarios durante un proceso de maquinado.

Para Jaber E. Abu, Al-Momani, Mohamed y Yamamoto [2], el tiempo de producción total para corte de una pieza utilizando máquinas CNC consiste en:

- **Tiempo de vuelo (Travel time).** Consiste en el tiempo para mover el *spindle* de la CNC entre operaciones.
- **Tiempo de conmutación (Switch time).** Es el tiempo requerido para cambiar la herramienta de corte para la siguiente operación.
- **Tiempo de corte (Cutting time).** Es el tiempo en el que la herramienta de corte avanza con una cierta velocidad de corte en el aire o en un material.

Existen diversas investigaciones que se concentran en el estudio del tiempo de corte, optimizando los parámetros antes mencionados, como lo es la geometría de la pieza, el material, el tipo de herramienta y el tipo de proceso; sin embargo, en esta investigación se analiza el tiempo de vuelo o *Travel Time* entre operaciones, por lo que la ruta denominada como CTTP o *Cutting Tool Travel Path* debe ser minimizada; el procedimiento de minimización de esta referencia se encuentra basada en Algoritmos Genéticos. Esta referencia forma parte del estado del arte de la presente investigación como referencia principal debido a que el objetivo de la presente investigación es precisamente minimizar el tiempo de vuelo entre operaciones utilizando algoritmos de optimización, específicamente *Colonia de Hormigas*.

Una de las publicaciones más recientes se ha registrado en el 2010, la cual los autores Adel T, Mohammed F. y Karim Hamza [3] contemplan la idea de la optimización del maquinado de orificios que se encuentran en forma rectangular considerándolos en forma matricial utilizando *Colonia de Hormigas*, la cual la ubica también como parte fundamental del estado del arte de la presente investigación; a diferencia de esta referencia, la presente investigación tiene como objeto de estudio optimizar cualquier arreglo de orificios incluyendo las formas rectangulares.

Si bien es cierto que la mayoría de paquetes comerciales de CAD/CAM incluyen las utilerías necesarias para la generación automática de código numérico, sin embargo, la mayoría de las rutas generadas no son lo suficientemente óptimas en el sentido de la minimización del tiempo de vuelo o *Travel time* que por consiguiente generan distancias muy grandes para el recorrido del *spindle* de una máquina CNC. Este problema es un claro ejemplo de aplicación al conocido *Problema del Agente Viajero* ya que el TSP es un problema del tipo NP Completo en la programación discreta en donde no se ha sabido que se genere una solución en un tiempo polinomial.

1.1 Objetivo General

Desarrollo e implementación de algoritmos de alto rendimiento que utilicen metaheurísticas basadas en colonias de hormigas para aplicaciones de tipo industrial.

1.1.1 Objetivos Específicos

- Optimizar el problema del Agente Viajero mediante Colonia de Hormigas utilizando computación paralela, para así reducir tiempos de ejecución del mismo.
- Desarrollo de plataforma de software para la optimización de procesos de manufactura industrial en máquinas de Control Numérico CNC reduciendo el tiempo de maquinado, específicamente del proceso de taladrado por medio de la Optimización por Colonia de Hormigas.
- Efectuar las comparaciones entre diversos paquetes comerciales y el algoritmo propuesto para la generación de código numérico.
- Análisis de los resultados relacionados a la eficiencia del algoritmo y la eficiencia en el maquinado.

1.1.2 Organización de la Tesis

El presente documento se encuentra organizado en cinco capítulos, en el capítulo 2 se presenta el marco teórico que sustenta el desarrollo de este trabajo de tesis, donde se tratan aspectos optimización y *metaheurísticas* así como la descripción de los tipos de problemas que se presentan en el área de la computación. Además se introduce el concepto de taladrado en esta sección y computación paralela.

En el capítulo 3 se describe la propuesta del presente trabajo de tesis el cual pretende desarrollar la descripción de la plataforma de software para la optimización de la trayectoria que tendría que seguir la herramienta de la máquina CNC mediante Colonia de Hormigas, *metaheurística* basada en el comportamiento de una colonia de hormigas natural, misma que su descripción se trata en este mismo capítulo y es punto importante para el presente trabajo. Además se describe el proceso por el cual el algoritmo interpreta un archivo *.DXF*, diseñando en algún paquete de CAD/CAM y es transformado en código G para efectos de fabricación. Cabe destacar que el presente trabajo tiene como objetivo específico el generar la trayectoria óptima para el proceso de taladrado, reduciendo tiempos de maquinado en general utilizando Colonia de Hormigas para la optimización del mismo. En este capítulo se muestra además las implementaciones relacionadas al cómputo de alto rendimiento, utilizando el concepto de paralelización para la búsqueda de la solución entre todas las hormigas artificiales.

En el capítulo 4 se presenta un análisis de resultados, destacando la importancia del algoritmo de optimización para la reducción de tiempos de fabricación. Finalizando el trabajo de investigación con el capítulo 5, presentando las conclusiones.

1.2 Aportaciones del trabajo

Existen diversos paquetes comerciales para diseño en CAD/CAM que generan la trayectoria para maquinado de piezas en 3D a partir de un diseño basado en CAD, sin embargo, durante la investigación se analizaron las trayectorias que se generan en estos paquetes y se ha podido concluir que son trayectorias no óptimas en relación al tiempo de manufactura; por lo que ha surgido la necesidad de optimizar dichas rutas para eliminar costos de operación y por supuesto tiempos de manufactura.

El presente trabajo presenta una alternativa para la generación de código numérico o también conocido como código G, optimizado en tiempos de manufactura utilizando la optimización mediante Colonia de Hormigas para la búsqueda de la ruta más corta. En el apartado 3 se presentará el diseño de la plataforma de software para la generación de Código G de la ruta optimizada para el procedimiento de perforación de un número de agujeros en una superficie plana de una placa de cualquier material.

Por otra parte, se ha realizado el registro de autor de la plataforma de software, mismo que se encuentra en trámite; además el artículo de divulgación fue aceptado al ser sometido para su publicación en *Springer Verlag* y en revista internacional.

Capítulo 2

Marco Teórico

2.1 Problemas P, NP y NP-Completos

De manera común se analizan algoritmos en donde el tiempo de ejecución puede ser expresado en términos de una función polinomial de un grado relativamente bajo. En este apartado analizaremos los problemas en donde no se han encontrado algoritmos eficientes para dar solución a estos mismos; es difícil imaginar si algún día existirá un algoritmo lo suficientemente eficiente que permita dar solución a este tipo de problemas.

Ejemplo 1. Sea $\prod$ cualquier problema. Decimos que existe un algoritmo de tiempo polinomial para dar solución al problema $\prod$ si existe un algoritmo para $\prod$ el cual su complejidad sea $O(n^k)$, donde n es el tamaño de entrada y k es un entero no negativo [4].

Como se puede observar, ningún problema del mundo real puede entrar dentro de esta categoría, ya que su solución requiere de una cantidad de tiempo que es medida en términos de funciones exponenciales y funciones hiper-exponenciales como lo es 2^n y $n!$. En las ciencias de la computación se ha referido a los problemas en donde existen algoritmos de tiempo polinomial como *tratables*, y a los que no existen algoritmos de este tipo se les ha denominado como *intratable.*

En este apartado estudiaremos una subclase de problemas intratables denominados como problemas NP-Completos. Este tipo de problemas tienen propiedades comunes, de las cuales si se llega a resolver un problema en tiempo polinomial, entonces todos los demás problemas se vuelven tratables. Muchos de estos problemas son naturales en el sentido que son aplicaciones del mundo real. Los tiempos de ejecución de los algoritmos que dan soluciones aproximadas a estos problemas varían de acuerdo a la cantidad de datos de entrada y pueden llegar a medir en términos de cientos o miles de años para resolverlos como se puede apreciar en la Tabla 2.1 [4].

Tabla 2.1 Tiempos de ejecución para diferentes tamaños de entrada. "nsec" se refiere a nanosegundos, "μ" a micro segundos y "cent" a siglos.

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3nsec	0.01μ	0.02μ	0.06μ	0.51μ	0.26μ
16	4nsec	$0.02\,\mu$	$0.06\,\mu$	$0.26\,\mu$	$4.10\,\mu$	$65.5\,\mu$
32	5nsec	$0.03\,\mu$	$0.16\,\mu$	$1.02\,\mu$	$32.7\,\mu$	4.29sec
64	6nsec	$0.06\,\mu$	$0.38\,\mu$	$4.10\,\mu$	$262\,\mu$	5.85 cent
128	$0.01\,\mu$	$0.13\,\mu$	$0.90\,\mu$	$16.38\,\mu$	0.01 sec	10^{20} cent
256	$0.01\,\mu$	$0.25\,\mu$	$2.05\,\mu$	$65.54\,\mu$	0.02 sec	10^{55} cent
512	$0.01\,\mu$	$0.51\,\mu$	$4.61\,\mu$	$262.14\,\mu$	1.07 sec	10^{135} cent
2048	$0.01\,\mu$	$2.05\,\mu$	$22.53\,\mu$	0.01sec	8.40 sec	10^{598} cent
4096	$0.01\,\mu$	$4.10\,\mu$	$49.15\,\mu$	0.02 sec	1.15 sec	10^{1214} cent
8192	$0.01\,\mu$	$8.19\,\mu$	$106.50\,\mu$	0.07 sec	1.15 min	10^{2847} cent
16384	$0.01\,\mu$	$16.38\,\mu$	$229.38\,\mu$	0.27 sec	1.22 hrs	10^{4913} cent
32768	$0.02\,\mu$	$32.77\,\mu$	$491.52\,\mu$	1.07 sec	9.77 hrs	10^{9845} cent
65536	$0.02\,\mu$	$65.54\,\mu$	$1048.6\,\mu$	0.07 min	3.3 days	10^{19709} cent
131072	$0.02\,\mu$	$131.07\,\mu$	$2228.2\,\mu$	0.29 min	26 days	10^{19438} cent
262144	$0.02\,\mu$	$262.14\,\mu$	$4718.6\,\mu$	1.15 min	7 months	10^{78854} cent
524288	$0.02\,\mu$	$524.29\,\mu$	$9961.5\,\mu$	4.58 min	4.6 years	10^{157808} cent
1048576	$0.02\,\mu$	$1048.60\,\mu$	$20972\,\mu$	18.3 min	37 years	10^{315634} cent

2.1.1 Clase P

Definición. Sea A un algoritmo para resolver $\prod$. Decimos que A es determinístico si sólo tiene una opción de solución en cada paso durante su ejecución, por lo que si A se ejecuta de nuevo n veces con las mismas entradas, su salida nunca cambiará.

La clase de problemas de decisión del tipo P consisten en todos aquellos problemas de decisiones en donde las soluciones del tipo *si/no* pueden obtenerse usando un algoritmo determinista que se ejecuta en un número polinomial de pasos, por ejemplo en $O(n^k)$ pasos, siendo k un entero positivo y n el tamaño de entrada.

Algunos ejemplos de problemas de tipo P son los siguientes:

- ORDENAMIENTO. Dada una lista de n enteros, ¿Se encontrarán ordenados de manera creciente?
- CONJUNTOS DISJUNTOS. Dado dos conjuntos de enteros, ¿Su intersección se encuentra vacía?
- CAMINO MÁS CORTO. Dado un grafo dirigido $G = (V, E)$ con pesos positivos en las aristas, dos vértices distintos $s, t \in V$ y un entero positivo k, ¿Existirá una ruta desde s hasta t la cual su longitud sea a lo mucho del tamaño de k?
- 2-COLORES. Dado un grafo no dirigido G, ¿Se puede colorear utilizando sólo dos colores y que no se repitan entre vértices?

A esta clase de problemas se le puede identificar si sus soluciones tienen un complemento, como lo es el ejemplo del problema de los colores, la pregunta complementaria sería: ¿No es coloreable?

2.1.2 Clase NP

La clase NP consiste en aquellos problemas $\prod$ por los que existe un algoritmo no determinístico A el cual será capaz de verificar su exactitud en tiempo polinomial; esto es, si la solución indica una respuesta viable, entonces existe una manera de verificar su solución en tiempo polinomial.

Un algoritmo *no determinista* consiste en dos fases:

1. **Fase de indagación**. En esta fase, una cadena de caracteres arbitraria es generada. Puede corresponder a una solución o no de la entrada. Esta posible solución no es la misma por cada ejecución del algoritmo, puede diferir de acuerdo al algoritmo no determinista. Sólo se requiere que esta cadena pueda generar una serie de pasos en tiempo polinomial como lo es en $O(n^i)$. En muchos de estos casos, esta fase puede ser completada en un tiempo lineal.

2. **Fase de verificación.** En esta fase, un algoritmo determinista verifica dos cosas. Primero revisa si la solución generada de la cadena y se encuentra en el formato apropiado; sino, el algoritmo se detiene con una solución negativa. Por otra parte, si y se encuentra en el formato adecuado, el algoritmo continúa revisando si existe una posible solución para una instancia x del problema. Si existe una solución a la instancia x, entonces el algoritmo se detiene y da una solución afirmativa, de lo contrario dará negativa. También se requiere que esta fase sea completada en un número de pasos polinomiales, por ejemplo en tiempo $O(n^j)$.

Definición. Sea A un algoritmo no determinista para un problema $\prod$. Decimos que A acepta una instancia I de $\prod$ si y sólo si en la entrada I existe la posibilidad de obtener una respuesta afirmativa. En otras palabras, A acepta I si y sólo si es posible que en alguna ejecución del algoritmo, la fase de verificación regrese una respuesta afirmativa. Si el algoritmo responde con una respuesta negativa, significa que A no acepta a su entrada.

El tiempo de ejecución de un algoritmo no determinista es simplemente la suma de los dos tiempos de ejecución de las fases:

$$O(n^i) + O(n^j) = O(n^k) \qquad (1)$$

para un entero positivo k.

En resumen, podemos distinguir de las dos clases de problemas por las siguientes propiedades:

- **P** es la clase de problemas de decisión en los cuales podemos decidir o resolver utilizando un algoritmo determinista que se ejecute en tiempo polinomial.
- **NP** es la clase de problemas de decisión en los cuales podemos verificar su solución utilizando un algoritmo determinista en tiempo polinomial.

Precisamente, la distinción de estas dos clases ha sido un problema en las ciencias de la computación, y es uno de los problemas que no se ha resuelto hasta la fecha; se ha denotado matemáticamente por: $P = NP?$ Naturalmente, cualquier problema P es también un problema NP, la pregunta que relaciona a P vs. NP trata si NP es un subconjunto de P, o si las clases son iguales. La importancia radica en que si las clases son iguales, entonces podemos resolver problemas que hasta ahora se han considerado como intratables. Si no son iguales, entonces los problemas NP Completos son problemas que

probablemente sean duros (*hard*). En la figura 2.1 se pueden apreciar los conjuntos de estas dos clases y sus relaciones.

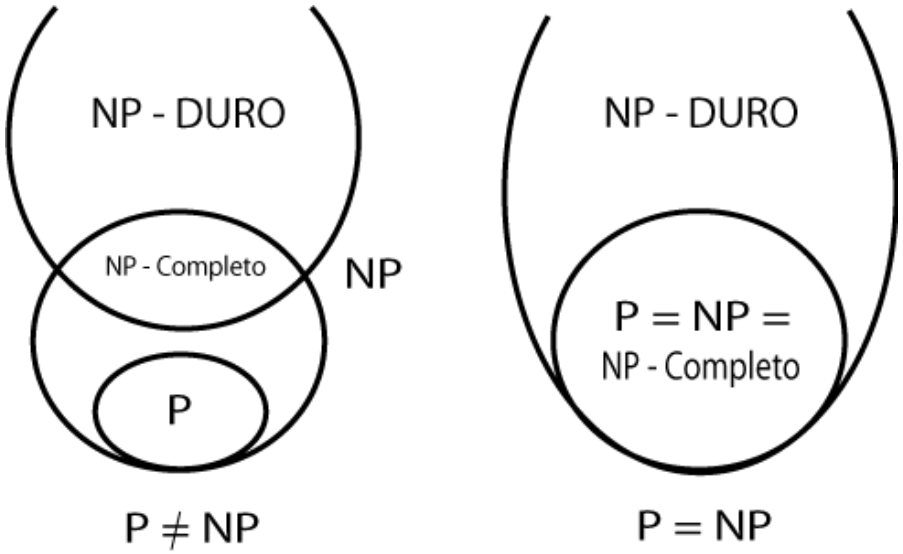

Figura 2.1 Diagrama de Euler para los conjuntos P, NP, NP-Completo y Duro.

2.1.3 Clase NP-Completo

El término "NP-Completo" denota la subclase de los problemas de decisión del tipo NP que son duros en el sentido de que si uno de ellos demuestra ser resuelto por un algoritmo determinista en tiempo polinomial, entonces todos los problemas en NP son tratables mediante un algoritmo determinista en tiempo polinomial, entonces se da la cuestión anteriormente mencionada como $NP = P$.

Definición. Sea $\prod$ y $\prod'$ dos problemas de decisión. Podemos decir que $\prod$ se reduce a $\prod'$ en tiempo polinomial, simbolizado como $\prod \alpha_{poly} \prod'$, si existe un algoritmo determinístico A. Cuando A se presenta con una instancia I del problema $\prod$, se transforma en una instancia I' del problema $\prod'$ tal que la respuesta de I es afirmativa si y

sólo si la respuesta I' es afirmativa. Esta transformación deberá ser en tiempo polinomial [4].

Para ejemplificar lo anterior, se procede a considerar los siguientes dos problemas:

1. *Problema de un ciclo Hamiltoniano.* Dado un grafo no dirigido $G = (V, E)$, ¿Existirá un ciclo Hamiltoniano?, un ciclo el cual visite cada vértice exactamente una vez.

2. *Problema del Agente Viajero.* Dado un conjunto de n ciudades con sus respectivas distancias entre ellas, y un entero k, ¿Existirá un tour de tamaño k? En este caso, un tour es un ciclo que visita cada ciudad exactamente una vez.

Se sabe que el problema de un Ciclo Hamiltoniano es un problema NP-Completo, utilizaremos este hecho para demostrar que el problema del Agente Viajero también es un problema del tipo NP-Completo. El primer paso en la demostración es mostrar que el TSP se encuentra dentro de los problemas NP, sabiendo que un algoritmo no determinista puede comenzar por generar una secuencia de ciudades y verificar después que esta secuencia sea una *ruta*. En este caso, el algoritmo continúa verificando si la distancia del tour es al menos del tamaño de k. El segundo paso es demostrar que un Ciclo Hamiltoniano se reduce a un TSP en tiempo polinomial, esto es:

$$\text{Ciclo Hamiltoniano } \alpha_{poly} \text{ TSP} \qquad (2)$$

Sea G una instancia arbitraria de un Ciclo Hamiltoniano. Se construye un grafo con pesos G' y un límite k tal que G tenga un Ciclo Hamiltoniano si y sólo si G' tenga un tour con distancia total de al menos k. Sea $G = (V.E')$ un grafo completo en el conjunto de los vértices V:

$$E' = \{(u,v) \mid u,v \in V\} \tag{3}$$

Entonces, asignamos una longitud a cada arista en E' como sigue:

$$l(e) = \begin{cases} l & \text{if} \quad \text{e} \in \text{E}, \\ n & \text{if} \quad \text{e} \notin \text{E}. \end{cases} \tag{4}$$

donde $n = |V|$. Finalmente, asignamos $k = n$. Como podemos observar, G tiene un ciclo Hamiltoniano si y sólo si G' tiene un tour de tamaño exactamente de n. La parte $k = n$ es parte de la reducción [4].

2.2 Metaheurísticas en los problemas de optimización combinatoria

2.2.1 Optimización

Matemáticamente hablando, se puede decir que la optimización en su forma general se puede describir de la siguiente manera:

$$\begin{array}{ll} \textit{minimizar} & f_i(x),(i = 1,2,...,M), \\ x \in \mathsf{R}^n & \\ \textit{sujeta} & \phi_j(x) = 0,(j = 1,2,...,J), \\ \textit{a} & \psi_k(x) \le 0,(k = 1,2,...,K), \end{array} \tag{6}$$

donde $f_i(x)$, $\phi_j(x)$, y $\Psi_k(x)$ son funciones del vector de diseño:

$$x = (x_1, x_2,..., x_n)^T. \tag{7}$$

Los componentes x_i de x son llamados como variables de decisión, y pueden ser reales continuas, discretas o de ambas formas. Las funciones $f_i(x)$ donde $i = 1,2,...,M$ son llamadas funciones objetivo, en el caso de que $M = 1$, existirá una sola función objetivo. La función objetivo se le llama muchas veces como la *función costo*.

El espacio que se extiende por las variables de decisión se le denomina como *espacio de diseño o espacio de búsqueda* R^n, mientras que el espacio formado por los valores de la función objetivo se le denomina como *espacio de soluciones o espacio de respuesta*. Las igualdades para ϕ_j y para las desigualdades para ψ_k son las *restricciones*. Si la desigualdad se denotara como ≥ 0 se formularía la optimización como un problema de maximización.

2.2.2 Clasificación de los problemas de optimización

La clasificación de los problemas de optimización aún no es bien definida y ha creado confusiones dentro de la literatura, sin embargo, una manera de clasificar dichos problemas es tomando en cuenta el número de objetivos, limitaciones, formas de las funciones, espacio de las funciones objetivo, tipo de las variables de diseño, incertidumbre en los valores y esfuerzo computacional, como se puede observar en la figura 2.2.

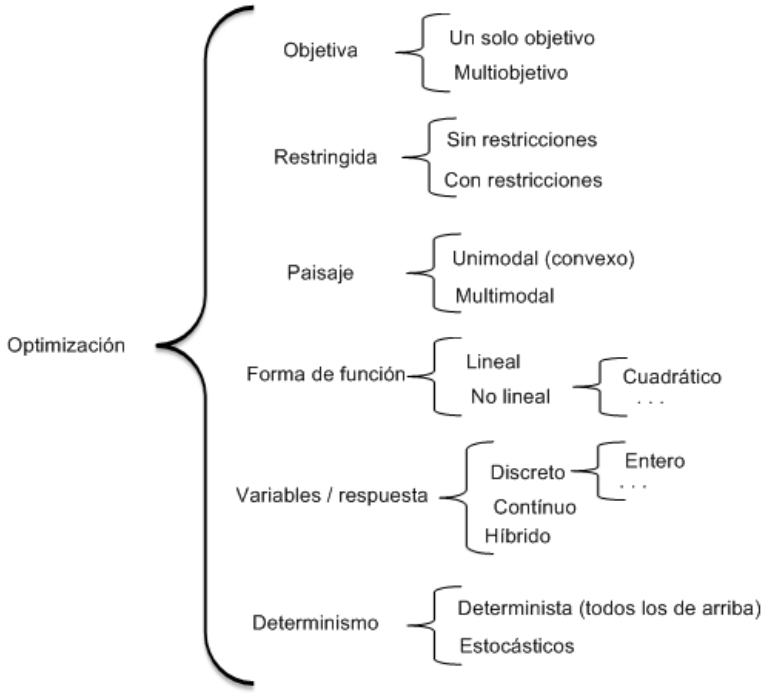

Figura 2.2 Clasificación de los problemas de optimización.

Si intentamos clasificar los problemas de optimización por el número de funciones objetivo, entonces existen dos clasificaciones: un sólo objetivo $M = 1$ y multiobjetivo $M > 1$, de igual manera mediante la clasificación basada en la forma de la función, podemos encontrar problemas lineales o no lineales, por su tipo de variables y respuestas dependiendo si son discretas, continuas o que presenten ambas características y por último por su grado de determinismo como lo son los problemas deterministas que son los antes mencionados y los estocásticos.

Todos los problemas de optimización son deterministas si para cada conjunto de variables de diseño, los valores de las funciones objetivo y las funciones de las restricciones se determinan exactamente. En el mundo real, sólo podemos saber algunos parámetros para un problema con incertidumbre. Si existe entonces incertidumbre y ruido en las

variables de diseño, en las funciones objetivo y en las restricciones, entonces la optimización se convierte en estocástica o una optimización robusta con ruido.

2.2.3 Algoritmos de Optimización

Podemos imaginar que la solución de un problema de optimización es como la búsqueda de un tesoro; imaginemos que tratamos de encontrar un tesoro en una montaña dentro de un límite de tiempo. La búsqueda se realiza normalmente de manera aleatoria que podría verse como una solución no muy eficiente como se espera. Otra manera de buscar la solución es conocer si el tesoro (la solución) se encuentra en la cresta, este escenario corresponde a una técnica denominada en inglés como *hill-climbing*. Si el tesoro se encuentra entre los extremos, normalmente no se sabe por dónde comenzar la búsqueda; en la mayoría de los casos buscaremos de manera aleatoria mediante pistas; si no se encuentra la solución en el lugar seleccionado, se procede a encontrar una siguiente solución viable y así sucesivamente hasta encontrar la aproximación o en este caso la solución definitiva.

La búsqueda aleatoria ha formado parte hoy en día en muchos algoritmos de optimización.

En general, los algoritmos de optimización se pueden dividir en dos categorías: los algoritmos deterministas y los algoritmos estocásticos. Los algoritmos deterministas siguen un procedimiento riguroso y su trayectoria y valores de las variables de diseño junto con las funciones son repetitivos. Por ejemplo, el escalar una montaña es considerado un algoritmo determinista ya que para el mismo punto de partida, seguirá la misma trayectoria cada vez que se ejecute el algoritmo.

Los algoritmos estocásticos presentan situaciones aleatorias. Un algoritmo genético es un algoritmo estocástico, las soluciones de la población inicial serán siempre distintas cada vez que se ejecute el algoritmo debido a que se utilizan mutaciones y reproducciones de manera aleatoria. También existen los algoritmos híbridos, como lo es escalar una montaña, puede uno comenzar con diferentes puntos de inicio o de partida. La ventaja de crear procedimientos aleatorios es evitar caer en optimizaciones locales y producir un estancamiento dentro del algoritmo. La figura 2.3 presenta la clasificación de los algoritmos de optimización.

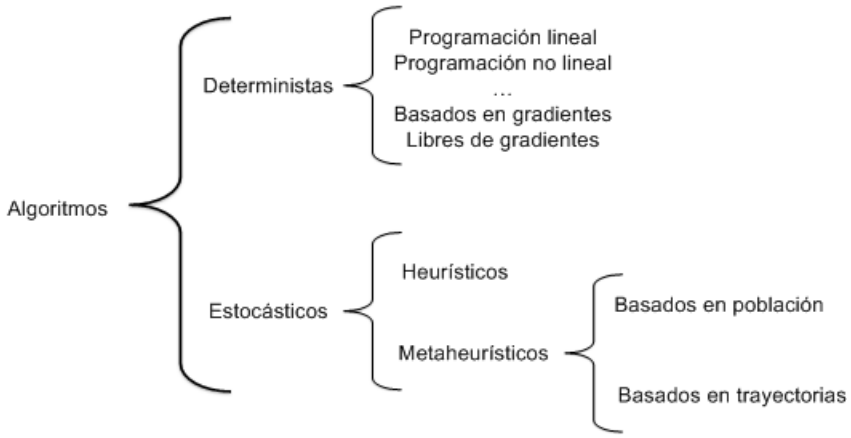

Figura 2.3 Clasificación de los algoritmos de optimización.

La mayoría de los algoritmos clásicos son deterministas, algunos de ellos utilizan programación lineal, otros utilizan métodos basados en gradientes como el conocido algoritmo *Newton-Raphson*.

Para los algoritmos estocásticos, se presentan en dos tipos, los *heurísticos* y los *metaheurísticos*, y su diferencia es mínima. *Heurística* significa *encontrar*, o *descubrir* mediante prueba y error [5]. La calidad de las soluciones de estos problemas pueden ser encontrados en una cierta cantidad de tiempo razonable, pero no se garantiza que la solución sea la óptima. Se espera que la mayoría de estos algoritmos trabajen la mayoría

del tiempo, pero no todo el tiempo. Después se ha llevado la optimización estocástica a la utilización del concepto de *Metaheurística*, que significa *más allá* o a un *nivel alto*, estos algoritmos han demostrado obtener mejores resultados que un algoritmo heurístico. Todos los algoritmos metaheurísticos utilizan un cierto tipo de procedimientos aleatorios y búsqueda local. Se sabe que no existe una definición formal para definir ambos tipos de algoritmos, sin embargo recientes investigaciones tienden a nombrar a todos los algoritmos estocásticos que presentan un grado de aleatoriedad y búsqueda local como *metaheurísticos*.

El grado de aleatoriedad permite realizar búsquedas locales para encontrar la optimización global del problema. La mayoría de los algoritmos metaheurísticos presentan buenas soluciones para la optimización global.

2.2.4 Metaheurísticas

La mayoría de los algoritmos metaheurísticos se inspiran de la naturaleza, basándose en el hecho de que la naturaleza ha encontrado a través de millones de años soluciones perfectas para la mayoría de los problemas que se ha encontrado.

Podemos aprender del éxito que la naturaleza ha tenido en la solución de sus problemas y podemos desarrollar algoritmos inspirados en la naturaleza ya sean heurísticos o metaheurísticos.

En los algoritmos metaheurísticos existen dos componentes principales, la selección de las mejores soluciones y el concepto de aleatoriedad. La selección de la mejor propuesta

permite encontrar un óptimo mientras que el grado de aleatoriedad permite evitar estancamientos en la solución.

Los algoritmos metaheurísticos se clasifican de diferentes maneras; una manera de clasificarlos es por medio de sus procedimientos si son basados en poblaciones o en trayectorias. Por ejemplo, los Algoritmos Genéticos son basados en poblaciones ya que utilizan un conjunto de cadenas, al igual que la Optimización por Partículas (PSO), el cual utiliza múltiples agentes o partículas. PSO también se le ha definido como un algoritmo basado en agentes. Por otra parte, el algoritmo de Temple Simulado utiliza un sólo agente o solución que se mueve en un espacio de diseño o espacio de búsqueda; un mejor movimiento o solución es siempre aceptado, mientras que una solución no tan viable puede ser aceptada mediante un grado de probabilidad.

2.2.4.1 Algoritmos inspirados en la naturaleza: metaheurísticas

Los sistemas naturales, especialmente las colonias, han inspirado a los científicos para descubrir nuevos algoritmos, los cuales son de ayuda en el área de la Inteligencia Artificial para el estudio de diversos fenómenos de la vida cotidiana, tales como la planeación, el aprendizaje, la toma de decisiones, la percepción, entre otros. A continuación se presentan algunas metaheurísticas que se han utilizado en la solución de problemas de combinatoria.

A. Algoritmos Genéticos

La esencia de los algoritmos genéticos se encuentra en la codificación de una función de optimización como un arreglo de bits o cadenas de caracteres que representan

los cromosomas, los operadores de manipulación y la selección de acuerdo a su función de aptitud con el propósito de encontrar una solución al problema de interés.

A una iteración la cual crea una nueva población es denominada como generación. La codificación de la función objetivo es usualmente en forma binaria. Por simplicidad, utilizamos cadenas de caracteres en forma binaria para codificar y decodificar. Los operadores genéticos utilizados son el cruzamiento, la mutación y la selección de la población [5].

B. Temple Simulado

La aplicación del Temple Simulado en problemas de optimización fue desarrollada originalmente por Kirkpatrick, Gelatt y Vecchi en 1983 [5]. El algoritmo de Temple Simulado es un método de búsqueda aleatorio que hace uso de las cadenas de Markov, el cual no solo acepta cambios que mejoran la función objetivo, también acepta cambios que no son ideales. En un problema de minimización, por ejemplo, cualquier movimiento bueno o cambios que reducen el valor de la función objetivo f será aceptada; sin embargo, algunos cambios que incrementan f serán aceptados con una probabilidad p [5].

C. Colonia de Abejas

Las abejas viven en colonias, ellas buscan y almacenan miel en la colonia que ellas mismas construyen. Las abejas se pueden comunicar por medio de feromona y por medio de un término que se le ha dado al movimiento de las abejas denominado como “waggle dance” en inglés [5]. Cuando las abejas encuentran una buena fuente de alimento y traen al panal el néctar, ellas se comunican por medio de las danzas de tipo *waggle*.

El algoritmo comienza con la ubicación de las abejas en diferentes fuentes de alimentación (flores) [6] para maximizar el total de néctar que se pueda obtener. La colonia tiene que optimizar la eficiencia de la recolección de néctar [7]. La ubicación de las abejas depende de diversos factores, tales como la concentración de néctar en la fuente de comida y la cercanía hacia el panal.

2.2.5 Optimización por Colonia de Hormigas para la solución del Problema del Agente Viajero

A. Problema del Agente Viajero

El problema del Agente Viajero es el problema en el cual un vendedor que, iniciando su recorrido desde su lugar de origen, requiere encontrar la ruta más corta para poder visitar un conjunto de ciudades cliente y de nuevo regresar a casa, visitando cada ciudad exactamente una vez.

Formalmente, el TSP (*Travelling Salesman Problem*) [8] puede ser representado con un grafo $G = (N, A)$ donde N es el conjunto de nodos que representan las ciudades, y A el conjunto de las aristas. A cada arista $(i, j) \in A$ se le asigna un valor o distancia (*length*) $d_i j$, el cual es la distancia entre las ciudades i y j, con $i, j \in N$ [8].

El objetivo principal en el TSP es encontrar la mínima distancia dentro de un circuito Hamiltoniano, donde un circuito Hamiltoniano es una ruta cerrada visitando cada uno de los $n = |N|$ nodos de G exactamente una vez. Entonces, una solución óptima al problema del TSP [9] es una permutación π de los nodos $1,2,...,n$ tal que la distancia $f(\pi)$ se encuentra dada por:

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \quad (8)$$

B. Algoritmos de Colonia de Hormigas para TSP

ACO puede ser aplicado para resolver el problema del Agente Viajero; el grafo de construcción $G = (C, L)$, donde el conjunto L se conecta con los componentes C, donde $C = N$ y $L = A$ del problema del grafo; el conjunto de estados del problema corresponde al conjunto de todas las rutas parciales posibles; y las restricciones Ω obligan a que las hormigas construyan solamente rutas o tours viables que corresponda a las permutaciones de los índices de las ciudades. Esto es siempre posible ya que la construcción del grafo es un grafo completo y cualquier trayectoria cerrada que visite todos los nodos sin que se repita ninguno, corresponde a una ruta viable.

En todos los algoritmos basados en ACO disponibles, los rastros de feromona se encuentran asociados con las aristas, entonces τ_{ij} se refiere a la deseabilidad de visitar la ciudad j después de haber visitado la ciudad i [10]. La información heurística es seleccionada como $\eta_{ij} = 1/d_{ij}$, [10] es decir la deseabilidad heurística de ir de la ciudad i a la ciudad j es inversamente proporcional a la distancia entre ambas ciudades. En el caso de que $d_{ij} = 0$ para alguna arista (i, j), la información heurística es 0 y η presenta un valor muy pequeño. Para propósitos de implementación, los rastros de feromona se recolectan dentro de una matriz de feromona y sus elementos son todas las η_{ij} [11].

Las rutas se construyen aplicando el siguiente procedimiento para cada hormiga:

1. Escoger, de acuerdo a algún criterio, una ciudad de inicio para ubicar a la hormiga inicial.
2. Utilizar los valores de feromona y valores heurísticos para construir una probabilidad para seleccionar las ciudades óptimas de manera iterativa, agregando ciudades que las hormigas no hayan visitado, hasta que todas las ciudades sean visitadas.
3. Volver a la ciudad inicial.

Después de que todas las hormigas hayan completado su tour, depositarán la feromona en los tours que hayan seguido. En algunos casos, antes de depositar la feromona en la matriz, las hormigas pueden mejorar su tour aplicando un procedimiento de búsqueda local; la figura 2.4 muestra el pseudocódigo general para resolver el problema del Agente Viajero por medio de Colonia de Hormigas.

```
Procedure ACO Metaheurística
    Inicializar parámetros, rastros de feromona
    while (no se cumpla la condición de paro) do
        Construir soluciones por hormiga
        Aplicar búsqueda local   %Opcional
        Actualizar rastro de feromona
    end
end
```

Figura 2.4 Procedimiento general de optimización mediante Colonia de Hormigas.

C. Sistema de Hormigas (Ant System)

El algoritmo de un Sistema de Hormigas (AS) lo constituyen dos fases: la primera consiste en la construcción de la solución de las hormigas y la segunda en la actualización del rastro de la feromona. En AS [10] una buena heurística para inicializar los rastros de feromona es fijar un valor inicial de acuerdo al número de hormigas; una estimación de este valor se puede obtener mediante la siguiente relación:

$$\forall(i, j), \tau_{i,j} = \tau_0 = m/C^{nm} \tag{9}$$

donde m es el número de hormigas, y C^{nm} es la longitud del tour generado por la heurística del vecino más cercano; este último puede ser cualquier algoritmo o procedimiento de construcción. Si los valores iniciales de feromona son muy altos, entonces muchas iteraciones se pierden esperando hasta que se evapore la feromona [10].

I. *Construcción de la ruta*

En AS, las hormigas artificiales m construyen una ruta para el TSP de manera concurrente. Inicialmente las hormigas se ubican de manera aleatoria en diversas ciudades. En cada paso de construcción, la hormiga k aplica una regla de probabilidad de decisión, denominada como *regla proporcional aleatoria*, para decidir qué siguiente ciudad visitar.

En particular, la probabilidad de que una hormiga k, estando en la ciudad i, seleccione ir a la ciudad j está dada por:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_j^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \qquad si\ j \in N_i^k, \tag{10}$$

donde $\eta_{ij} = 1/d_{ij}$ es el valor heurístico que está disponible de manera *apriori*, α y β son dos parámetros que determinan la influencia relativa del rastro de feromona y el valor heurístico, y N_i^k es la vecindad viable para la hormiga k cuando se encuentra en la ciudad i, esto es, el conjunto de ciudades que la hormiga k no ha visitado aún (la probabilidad de seleccionar una ciudad fuera de N_i^k es 0).

Por la regla de probabilidad, la probabilidad de seleccionar una arista (i, j) en particular se incrementa en relación al valor asociado con el rastro de feromona τ_{ij} y el valor heurístico η_{ij}. El rol de los parámetros α y β es como sigue: Si $\alpha = 0$, las ciudades más cercanas serán las que se visiten primero, esto corresponde a un algoritmo estocástico clásico conocido como *Greedy* con múltiples puntos de inicio ya que las hormigas iniciarían de manera aleatoria. Si $\beta = 0$, solamente la amplificación de la feromona trabaja, esto es, solamente el parámetro de feromona es utilizado, sin utilizar el valor heurístico [11] [12]. Estos resultados presentan soluciones pobres; las recomendaciones para los parámetros de control se muestran en la Tabla 2.2.

Tabla 2.2 Parámetros recomendados para diferentes algoritmos de ACO.

Algoritmo ACO	α	β	ρ	m	τ_0
AS	1	2 – 5	0.5	n	m/C^m
EAS	1	2 – 5	0.5	n	$(e+m)/\rho C^m$
AS_{rank}	1	2 – 5	0.1	n	$0.5r(r-1)/\rho C^m$
$\mathcal{M}\ \mathcal{M}AS$	1	2 – 5	0.02	n	$1/\rho C^m$
ACS	–	2 – 5	0.1	10	$1/n\rho C^m$

II. *Actualización del rastro de feromona*

Después de que todas las hormigas han construido sus rutas, los rastros de feromona son actualizados. Esto se realiza reduciendo el valor de feromona en todas las aristas de acuerdo a un factor constante y después agregando feromona en las aristas por las cuales las hormigas han cruzado [13]. La evaporación de la feromona [10] se implementa mediante la siguiente asignación:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \qquad \forall (i,j) \in L, \tag{11}$$

donde $0 < \rho \leq 1$ es la tasa de evaporación de la feromona. El parámetro ρ es utilizado para evitar la acumulación ilimitada de los rastros de feromona y permite que el algoritmo no se "olvide" decisiones malas que anteriormente haya tomado; si una arista no es elegida por las hormigas, el valor asociado de feromona se reduce de manera exponencial de acuerdo al número de iteraciones. Después de la evaporación, todas las hormigas depositan feromona en las aristas que hayan recorrido en su ruta:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \qquad \forall (i,j) \in L, \tag{12}$$

donde $\Delta\tau_{ij}^{k}$ es la cantidad de feromona que la hormiga k deposita en las aristas que ha visitado. Se define de la siguiente manera:

$$\triangle_{ij}^{k} = \begin{cases} {}^{1}\!/_{C^k}, & \text{si la arista } (i,j) \in T^k \\ 0, & \textit{de otro modo}; \end{cases} \tag{13}$$

donde C^k es el tamaño de la ruta T^k construida por la hormiga k; se calcula mediante la suma de todas los tamaños de las aristas que pertenecen a la ruta T^k.

La optimización por Colonia de Hormigas es entonces considerada como una metaheurística debido a que presenta la característica de encontrar mejores soluciones en cada iteración, para así encontrar el óptimo; además como se ha visto, también presenta un grado de aleatoriedad para evitar un estancamiento.

2.3 Conceptos de cómputo paralelo

El uso de la programación y arquitecturas paralelas es esencial para la simulación y solución de problemas en la práctica computacional moderna. El rápido progreso en las arquitecturas de los microprocesadores, las tecnologías de interconexión y el desarrollo de software, han sido influencia directa en el desarrollo de tecnologías basadas en cómputo paralelo y distribuido.

2.3.1 Programación secuencial y paralela

Tradicionalmente, se han diseñado algoritmos de manera secuencial, esto significa que se ejecutan en una sola computadora que cuenta con una Unidad de Procesamiento Central (CPU). El problema se divide en instrucciones que se ejecutan de modo secuencial o serial, de manera que se ejecuta una instrucción por un tiempo determinado, de acuerdo a los ciclos máquina que le haya tomado dicha instrucción; al finalizar de ejecutarse la

misma, se procede con la siguiente instrucción y así sucesivamente [16]. Esto se puede apreciar en la Figura 2.5.

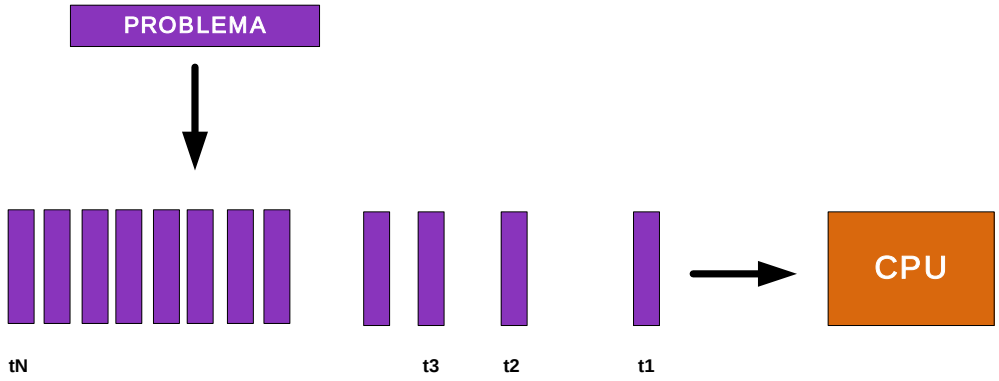

Figura 2.5 Ejecución y distribución de instrucciones en un algoritmo secuencial.

Cuando un problema puede dividirse en tareas independientes, se dice que se puede implementar utilizando cómputo paralelo, que es el uso simultáneo de recursos de hardware para dar solución a un problema; esto significa que se pueden ejecutar las tareas de forma independiente en cada unidad de procesamiento o CPU dividiendo el problema en varios bloques que se ejecutarán de manera concurrente, y cada uno de ellos se dividen en instrucciones que se ejecutarán en cada CPU, como se puede observar en la Figura 2.6.

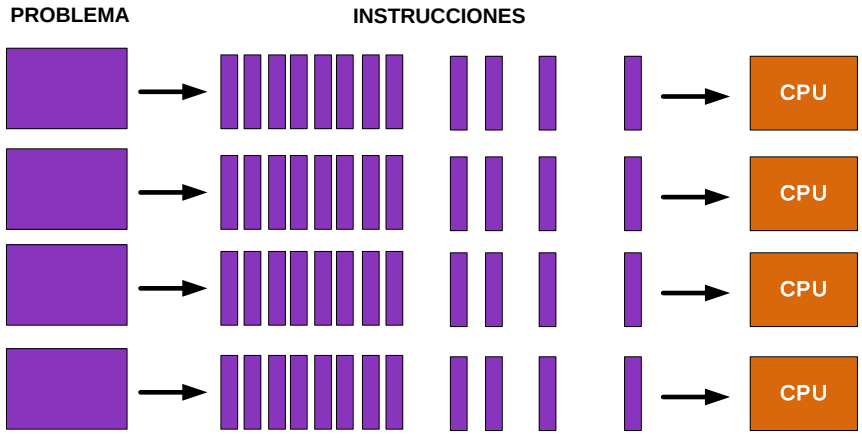

Figura 2.6 Ejecución y distribución de instrucciones en un algoritmo concurrente.

2.3.2 Arquitecturas paralelas

Las arquitecturas multinúcleo han comenzado una nueva era de la computación [17], el problema radica en cómo ejecutar programas secuenciales de manera eficiente y confiable en tecnologías multinúcleo. Las arquitecturas multinúcleo proveen de mejor poder de cómputo y eficiencia en potencia que en tecnologías monolíticas. Se espera que las tecnologías multinúcleo tiendan aún a incrementarse de acuerdo a la Ley de Moore, por lo menos una década más [17].

Sin embargo, el acceso limitado a memoria se espera que sea uno de los problemas en las tecnologías con diversos núcleos en un chip, debido a la jerarquía de acceso a la memoria caché.

Un ejemplo de una arquitectura novedosa es la Arquitectura de Motor Cell de Banda Ancha o *Cell Broadband Engine Architecture* desarrollada en conjunto por IBM, Sony y Toshiba [18]. Es un chip multinúcleo que difiere significativamente de multiprocesadores convencionales. La celda BE es un chip heterogéneo de más de de 200GFlop por chip. El mercado principal inicialmente fue la consola de videojuegos *Sony's PlayStation 3* [19].

El número de núcleos en un chip continúa incrementándose: existen actualmente reportes de 64 [20] y hasta 80 [21] núcleos en un solo chip. Todos estos nuevos diseños confirman que se requerirá de nuevas herramientas, nuevos algoritmos y una nueva manera de visualizar la programación convencional.

Los dispositivos NoC o Network – on – Chip constituyen un nuevo paradigma en la arquitectura en las redes de comunicación [22]. Se ha dado por el avance en la nanotecnología en CMOS y su función es proveer datos entre el nodo fuente y el destino dentro de un sistema de gran escala de integración (VLSI – *Very Large Scale Integration*).

Las compuertas lógico – programables FPGAs [23] son una nueva clase de bloques para sistemas de cómputo paralelo masivo. Son dispositivos programables que están interconectados por medio de componentes lógicos o compuertas lógicas. Debido a su flexibilidad, el uso de los dispositivos FPGAs han sido utilizados como hardware de propósito específico, además se pueden reconfigurar para ser utilizados como arquitecturas de cómputo paralelo [24]. Como se han podido reconfigurar para estos propósitos, han logrado alcanzar un perfil en la computación moderna conocida como *High – Performance Computing (HPC)* [25].

Estos componentes programables pueden ser utilizados en aplicaciones como en procesamiento de señales [26] o en criptografía [27].

Otra de las nuevas tendencias en las tecnologías multinúcleo son las denominadas como GPUs o por sus siglas en inglés *Graphics Processing Unit* [28] en un modelo de programación que debe contemplar el uso de un CPU y un GPU en conjunto. La parte secuencial se ejecuta en el procesador, y la parte computacionalmente intensiva se ejecuta en un GPU.

2.3.3 Distribución de memoria

A. Memoria compartida

La principal característica de este tipo de arquitectura está dada por el uso compartido de la memoria que presenta cada procesador, esto es si una localidad de memoria es afectada, esta modificación será visible por todos los procesadores. Esta arquitectura se puede dividir en dos, *UMA* y *NUMA* [15][16].

UMA (Uniform Memory Access). Se distingue por la simetría en los procesadores, de igual manera existe un mismo acceso a memoria con igual tiempos de acceso. En la figura 2.7 se muestra la distribución de memoria mediante UMA.

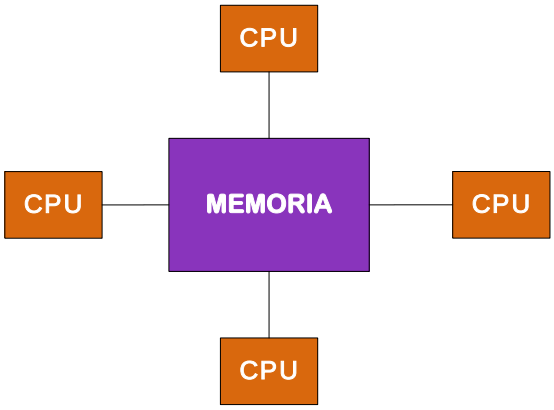

Figura 2.7 Memoria compartida UMA.

NUMA (Non-Uniform Memory Access). En este tipo de arquitectura se puede diseñar uniendo dos o más Multiprocesadores simétricos. En este caso, no todos los

procesadores tienen el mismo tiempo de acceso a todas las memorias. El acceso a memoria a través del bus es mucho más lenta que de manera independiente. En la figura 2.8 se muestra esta arquitectura.

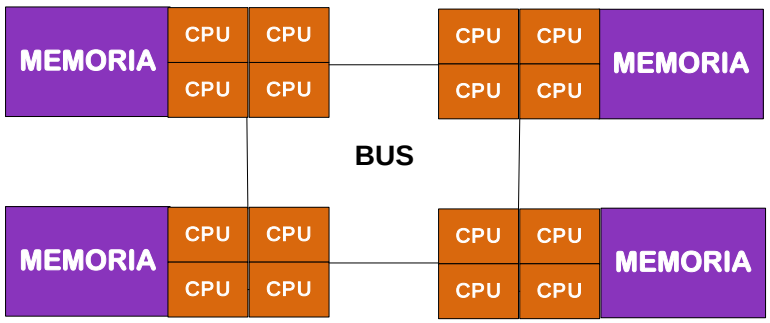

Figura 2.8 Memoria compartida NUMA.

B. Memoria distribuida

Para que un sistema presente este tipo de arquitectura deberá de contar una red de comunicación para interconectar procesadores y su memoria respectiva, como se puede observar en la figura 2.9 [16].

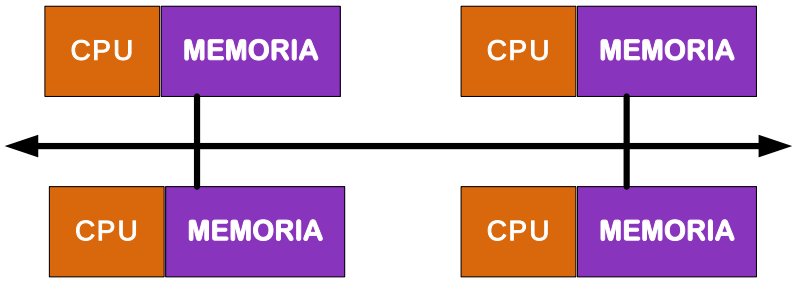

Figura 2.9 Memoria distribuida.

Todos los procesadores cuentan con su propia memoria local. La dirección de memoria en un procesador no mapea a otro procesador, por lo que no existe el concepto de direcciones de memoria globales entre todos los procesadores.

Debido a que cada procesador cuenta con su propia memoria local, operan de manera independiente. Los cambios que se realicen en una memoria local no presentan efecto alguno en otra memoria de otros procesadores en la red. Cuando un procesador requiere accesar a un dato en otro procesador, usualmente es tarea del programador indicar cuándo y cómo los datos se mantendrán en comunicación, además de la sincronización de las tareas.

2.3.4 Rendimiento de los sistemas paralelos

Una forma de medir la calidad de un sistema de alto rendimiento consiste en comparar la velocidad conseguida con el sistema paralelo (n procesadores) con la velocidad conseguida con un solo procesador. Por ello, se define la *ganancia de velocidad* o *aceleración (speedup)* de un sistema de n procesadores como:

$$S\left(N_p\right) = \frac{T\left(N_p=1\right)}{T\left(N_p\right)}, \tag{14}$$

donde $T\left(N_p = 1\right)$ es el tiempo empleado para ejecutar el proceso en un solo procesador y $T\left(N_p\right)$ es el tiempo empleado para ejecutarlo en un sistema paralelo con N_p procesadores [40]. En condiciones ideales, $T\left(N_p\right) = 1/N_p$ con lo que, en esas mismas condiciones, la ganancia de velocidad será:

$$S_{ideal}\ (N_p) = \frac{T(N_p=1)}{T(N_p)} = \frac{1}{1/N_p} = N_p \tag{15}$$

Una forma de medir el rendimiento del sistema, será comparar la ganancia de velocidad del sistema con la ganancia de velocidad ideal, a esta medida se le denomina como *eficiencia* que indica la medida en que se aprovechan los recursos y está dada por:

$$E\ (N_p) = \frac{S\ (N_p)}{S_{ideal}\ (N_p)} = \frac{T(N_p=1)/T(N_p)}{N_p} = \frac{T(N_p=1)}{N_p T(N_p)} = \frac{S\ (N_p)}{N_p} \tag{16}$$

2.4 Máquinas de Control Numérico por Computadora

La tecnología de Control Numérico se ha establecido a mediados del siglo XX, precisamente desde 1952 por la Fuerza Aérea de los Estados Unidos y en el Massachussetts Institute of Technology (MIT) en Cambridge, MA, USA. No se aplicó en la producción masiva de manufactura de piezas hasta en 1960. El *boom* del control numérico aplicado en una CNC (Computer Numerical Control) fue en el año 1972, y una década después con la introducción de microcomputadoras [29].

En el área de manufactura, y particularmente en el área de manufactura con metales, la tecnología de Control Numérico ha causado una revolución, inclusive antes de la llegada de las computadoras.

2.4.1 Definición de Control Numérico

De manera formal se puede decir que el Control Numérico es la operación de herramientas de máquina por medio de instrucciones o códigos de operación específicos hacia el sistema de control de la máquina [29].

Las instrucciones son la combinación de letras, dígitos y algunos símbolos como el punto decimal, el signo de porcentaje y los símbolos de paréntesis. Todas las instrucciones son escritas en un orden lógico y de una forma predeterminada. El conjunto de todas las instrucciones necesarias para maquinar una parte se le denomina como un programa NC (NC program). Este programa puede ser almacenado para aplicaciones futuras y utilizarlo repetidamente para la producción de múltiples piezas de las mismas dimensiones y características.

2.4.2 Tecnología NC y CNC

Existe una diferencia ente los términos NC y CNC. NC se le ha referido como a la tecnología más antigua de Control Numérico mientras que la abreviatura CNC se le ha denotado por sus siglas en inglés como Computer Numerical Control. CNC es la abreviatura que más se utiliza hoy en día [29].

La tecnología NC utiliza funciones lógicas la cuales se construyen de manera permanente por medio de cableado en la unidad de control. Estas funciones no pueden ser modificadas por el programador o el operador de la máquina. Máquinas modernas como la CNC utilizan microprocesadores, (una computadora interna), mismo que contiene los

registros de memoria que almacenan una serie de rutinas capaces de manipular las funciones lógicas. Esto significa que el programador o el operador de la máquina pueden modificar el programa en la unidad de control (en la máquina).

2.4.3 Planeación de un programa en CNC

Los pasos que se requieren para la planeación de un programa se definen por la naturaleza del trabajo [29]. No existe una fórmula para todos los trabajos, pero si existen procedimientos que se deben considerar:

- Información inicial: Características de las herramientas
- Complejidad de la parte
- Manuales y programación computarizada
- Procedimiento de programación típico
- Dibujo de la parte
- Especificaciones de los materiales
- Secuencia de maquinado
- Selección de herramientas
- Ajuste de la parte

Características de las herramientas

Es aquí donde el programador se concentra en un tipo de herramienta para usarse en un sistema CNC en particular. Cada parte debe estar montada en una fixtura, por lo

que la máquina CNC debe ser del tamaño adecuado para poder maquinar la parte sobre la fixtura diseñada.

Complejidad de la parte

Aquí es donde el programador se debe hacer las siguientes preguntas, ¿Qué tan complejo será programar manualmente la parte diseñada?, ¿Cuáles son las capacidades de las máquinas?, ¿Cuáles son los costos?

Para la programación de partes sencillas, esto es, una parte sencilla presenta una geometría regular, por lo que la programación manual es adecuada, ya que se siguen las coordenadas de manera simétrica en un programa NC mediante los comandos adecuados.

Para la programación de partes complejas, las cuales presentan formas irregulares, y por consiguiente formas asimétricas, la programación manual se vuelve compleja, por lo que en la actualidad se han desarrollado tecnologías asistidas por computadora, las cuales para el diseño de la misma pieza se han desarrollado paquetes comerciales CAD que por sus siglas en inglés se refiere al Diseño Asistido por Computadora; y para la manufactura de la pieza, lo cual requiere de la generación de código automático, se han desarrollado paquetes comerciales CAM o Manufactura Asistida por Computadora para la generación de código automático cuando una pieza presenta cierta complejidad para la programación manual.

CAD/CAM y CNC

La necesidad para mejorar la eficiencia y la calidad en la programación en CNC ha sido la principal razón por la cual se han desarrollado estos paquetes comerciales de diseño. CAD (*Computer Aided Design*) referente al diseño visual de la pieza y CAM (*Computer Aided Manufacturing*) referente al diseño del programa de maquinado conforman lo que ahora se le conoce como CIM (*Computer Integrated Manufacturing*) o Manufactura Integrada por Computadora.

- Integración

La palabra clave en el acrónimo CIM es *Integración*. Significa poner los elementos de manufactura relacionados y trabajar con ellos como una única unidad de manera más eficiente. Una de las reglas más importantes en CAD/CAM es "¡Nunca trabajes dos veces!" ("Never do anything twice!") [33] [34]. Cuando un dibujo es hecho en un software de CAD como AutoCAD®, SolidWorks®, Pro/E®, entre otros, y se realiza de nuevo el diseño en un software CAM como MasterCAM®, es trabajar dos veces. Esta duplicidad fomenta la generación de errores. Para evitar estos errores, se han creado métodos en la mayoría de los paquetes comerciales de CAD que incorporan métodos de transferencia del diseño al sistema de CAM que será utilizado.

Los métodos típicos de transferencia de diseños de CAD hacia CAM se hacen mediante archivos DXF o IGES. Estos acrónimos consisten en lo siguiente: DXF como *Data Exchange Files or Drawing Exchange Files* y los archivos IGES como *Initial Graphics Exchange Specificacion* [29].

Una vez que la transferencia del diseño desde el sistema CAD hacia CAM se haya realizado, sólo se requiere la generación de la trayectoria de la herramienta o *toolpath*, utilizando un post procesador (una rutina especial del programa) el software de computadora genera el programa que será ejecutado en la máquina CNC.

Procedimiento típico de programación

La planeación de un programa CNC no es diferente a cualquier otra planeación, debe ser ejecutada de una manera lógica y metodológica. Las primeras decisiones se relacionan con las tareas a realizarse y los objetivos a alcanzar. Las siguientes decisiones se basan en el cómo se llevará a cabo la tarea para alcanzar el objetivo de una manera eficiente y segura. Los siguientes pasos son el procedimiento típico de programación CNC:

- Estudio de la información inicial (Dibujo y métodos)
- Evaluación del material
- Especificaciones de las herramientas
- Características de un sistema de control
- Selección de las herramientas y el ordenamiento de las mismas
- Ajuste de las partes
- Datos técnicos (velocidades, avances o *feedrates*, etc.)
- Determinación de la trayectoria de la herramienta o *toolpath*
- Cálculos matemáticos
- Programación y preparación para la transferencia hacia la máquina CNC
- Pruebas y depuración del programa
- Documentación del programa

Estructura de un programa CNC

Un programa CNC se compone de una serie de instrucciones secuenciales relacionadas al maquinado de la parte. Cada instrucción es especificada con un formato que el sistema CNC pueda aceptar, interpretar y procesar [30]. Cada instrucción también incluye las especificaciones de la herramienta.

Existen cuatro términos básicos en la programación CNC:

Caracter → Palabra → Bloque → Programa

- *Caracter.* Es la unidad mínima en un programa CNC. Puede tener alguna de las siguientes tres formas:
    - Dígito
    - Letra
    - Símbolo

  Esta combinación se le denomina como entradas alfanuméricas.

- *Palabra.* Una palabra del programa es la combinación de caracteres alfanuméricos, creando una sola instrucción de control. Palabras típicas pueden indicar:
    - Posición de los ejes
    - Avance o *Feedrate*
    - Velocidad o *Speed*
    - Comandos de preparación
    - Funciones misceláneas

- *Bloque.* Un bloque es un conjunto de instrucciones para el sistema CNC. Cada línea o bloque de secuencia contiene una o varias palabras y cada palabra contiene dos o más caracteres.

- *Programa.* Un programa inicia con un número o una identificación similar, seguida por el conjunto de bloques y termina con un comando de paro o símbolo de aro como el signo de porcentaje %. Se pueden poner comentarios entre las líneas por medio de paréntesis, de la siguiente manera: (comentario).

Formatos de programación

Los formatos de programación se dividen de la siguiente manera:

- Formato secuencial de Espaciado (Tab Sequencial Format) para NC únicamente.
- Formato Fijo (Fixed Format) para NC únicamente.
- Formato de palabras de dirección (Word Address Format) para NC o CNC.

Los dos primeros formatos desaparecieron en 1970 y en la actualidad se utiliza el formato de palabras de dirección [30].

Formato de palabras de dirección (Word Address Format)

Este formato se basa en la combinación de una letra y varios dígitos. En algunas aplicaciones se puede complementar mediante un símbolo como un – o un + o un punto decimal. [30] Cada letra, dígito o símbolo representa un carácter en el programa y en la

memoria de control. La letra representa la dirección a un registro en específico de la memoria seguido por un dato numérico con o sin símbolo.

Palabras típicas en la programación de CNC son las siguientes:

```
G01 M30 D25 X5.75 N105 H01 Y0 S2500
Z-5.14 F12.0 T0505 T05 /M01 B180.0
```

No existen espacios dentro de las palabras y la dirección siempre debe escribirse primero, por ejemplo X5.75 es correcto mientras que 5.75X no lo es. El dato indica la asignación numérica de la palabra. Este valor depende de la dirección; puede representar un número de secuencia N, un comando de preparación G, una función miscelánea M, y un registro de offset como D o H, una palabra de coordenadas como X, Y o Z, la función de avance o *fedérate*, la función *spindle*, la función de la herramienta o *tool* T, etc (Ver apéndice I).

Algunos ejemplos del significado de algunas palabras se muestran a continuación:

`G01` Comando de preparación

`M30` Función miscelánea

`D35` Número de selección de offset

`X5.75` Palabra de coordenada (valor positivo)

`N105` Número de secuencia (número de bloque)

`H01` Número de offset del tamaño de herramienta

`Y0` Palabra de coordenada (valor cero u origen)

`S2500` Función de la velocidad del *spindle*

`Z-5.14` Palabra de coordenada (valor negativo)

`F12.0` Función de *feedrate* o avance

`T0505` Función de herramienta

`T05` Función de herramienta

`/M01` Función miscelánea con símbolo de omisión

`B180.0` Función de tablas indexadas

Un ejemplo de bloque se muestra a continuación:

```
N25 G90 G00 X13.0 Y4.6 M08
```

donde

N25 es el número de secuencia o bloque

G90 Modo absoluto

G00 Modo de rápido posicionamiento

X13.0 Y4.6 Coordenadas de ubicación

M08 Encendido de refrigerante (*coolant*)

Estructura típica de un programa

```
O0701 (ID MAX 15 CHARS)                          (PROGRAM NUMBER AND ID)
(SAMPLE PROGRAM STRUCTURE)                       (BRIEF PROGRAM DESCRIPTION)
(PETER SMID - 07 - DEC - 01)                     (PROGRAMMER AND DATE OF LAST REVISION)

N1 G20                                           (UNITS SETTING IN SEPARATE BLOCK)
N2 G17 G40 G80 G49                               (INITIAL SETTINGS AND CANCELLATIONS)
N3 T01                                           (TOOL T01 INTO WAITING POSITION)
N4 M06                                           (T01 INTO SPINDLE)
N5 G90 G54 G00 X.. Y.. S.. M03 T02               (T01 RESTART BLOCK - T02 INTO WAITING POSITION)
N6 G43 Z2.0 H01 M08                              (TOOL LG OFFSET - CLEAR ABOVE WORK - COOLANT ON)
(N7 G01 Z-.. F..)                                (FEED TO Z DEPTH IF NOT A CYCLE)
( ----- CUTTING MOTIONS WITH TOOL T01 ---- )

...
N33 G00 G80 Z2.0 M09                             (CLEAR ABOVE PART - COOLANT OFF)
N34 G28 Z2.0 M05                                 (HOME IN Z ONLY-SPINDLE OFF)
N35 M01                                          (OPTIONAL STOP)

N36 T02                                          (TOOL T02 INTO WAITING POSITION - CHECK ONLY)
N37 M06                                          (T02 INTO SPINDLE)
N38 G90 G54 G00 X.. Y.. S.. M03 T03              (T02 RESTART BLOCK - T03 INTO WAITING POSITION)
N39 G43 Z2.0 H02 M08                             (TOOL LG OFFSET - CLEAR ABOVE WORK - COOLANT ON)
(N40 G01 Z-.. F..)                               (FEED TO Z DEPTH IF NOT A CYCLE)
( ----- CUTTING MOTIONS WITH TOOL T02 ---- )

...
N62 G00 G80 Z2.0 M09                             (CLEAR ABOVE PART - COOLANT OFF)
N63 G28 Z2.0 M05                                 (HOME IN Z ONLY - SPINDLE OFF)
N64 M01                                          (OPTIONAL STOP)

N65 T03                                          (TOOL T03 INTO WAITING POSITION - CHECK ONLY)
N66 M06                                          (T03 INTO SPINDLE)
N67 G90 G54 G00 X.. Y.. S.. M03 T01              (T03 RESTART BLOCK- T01 INTO WAITING POSITION)
N68 G43 Z2.0 H03 M08                             (TOOL LG OFFSET - CLEAR ABOVE WORK - COOLANT ON)
(N69 G01 Z-.. F..)                               (FEED TO Z DEPTH IF NOT A CYCLE)
( ----- CUTTING MOTIONS WITH TOOL T03 ---- )

...
N86 G00 G80 Z2.0 M09                             (CLEAR ABOVE PART - COOLANT OFF)
N87 G28 Z2.0 M05                                 (HOME IN Z ONLY - SPINDLE OFF)
N88 G28 X.. Y..                                  (HOME IN XY ONLY)
N89 M30                                          (END OF PROGRAM)
%                                                (STOP CODE - END OF FILE TRANSFER)
```

Figura 2.10 Estructura típica de un programa CNC.

2.5 Introducción al procedimiento de Taladrado mediante CNC

En las operaciones de maquinado, el material, especialmente metales, muchas veces se ve forzado de modo tal que se crean rupturas en el material por las herramientas de corte. Durante la acción de corte, el metal se pone en contacto con las puntas de la herramienta de corte (broca), y se remueve constantemente material para obtener un orificio [31].

La principal herramienta de corte para el procedimiento de taladrado es la broca (*drill*). Taladrado es el procedimiento de producir orificios circulares por medio de brocas. En la figura 2.11 se muestra el diagrama de una broca genérica.

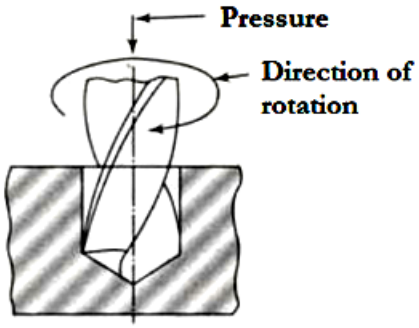

Figura 2.11 Diagrama de una broca en acción.

En el taladrado gira, generalmente, la herramienta. Durante la operación avanza en dirección axial contra la pieza que permanece en reposo. En algunos casos especiales el movimiento de corte lo realiza la pieza dotada de movimiento de rotación como pasa, por ejemplo, cuando se taladra en el torno con la broca en reposo. En algunas pequeñas taladradoras de sobremesa la pieza se avanza contra la broca y elevando para ello la mesa.

En las máquinas de taladrado profundo giran la herramienta y la pieza en sentidos opuestos. La herramienta realiza en este caso el movimiento de avance [31].

2.5.1 Herramientas de Taladrado

Las herramientas de taladrado presentan la forma de una cuña como la forma fundamental del filo en todas las variantes. Las principales herramientas de taladrado son la broca espiral, la broca pequeña, la broca para taladrar cañones, la broca de profundidad y la barra o eje de taladrar o barrenar. Específicamente trataremos la broca espiral como principal herramienta de corte para el presente trabajo.

Broca espiral

La broca de espiral de la Figura 2.12 es la herramienta de taladrado más usada. Sus ventajas especiales son las siguientes:

- Ángulos favorables en los filos
- Diámetro constante en el afilado
- Buena posibilidad de sujeción
- Buena guía dentro de la pieza
- Expulsión automática de las virutas al exterior del taladro

Como material para las brocas espirales se emplea acero de herramientas sin alear o aleado. Para materiales muy duros, (fundición dura, acero duro al manganeso), para

materiales sintéticos y para materiales prensados, ebonita, etc., se utilizan brocas con filos postizos de metal duro o también brocas totalmente de metal duro [31].

Figura 2.12 Designaciones en la broca espiral.

La broca espiral posee dos ranuras helicoidales para viruta situadas en posición opuesta. Están obtenidas del material macizo mediante fresado o mediante rodamiento. Las brocas hasta unos 13mm de diámetro son obtenidas del material macizo mediante muelas de esmeril de perfil adecuado. Las ranuras fresadas se esmerilan además cuando se trata de brocas muy valiosas [31].

El ángulo de la espiral de la broca en la Figura 2.13 es el que forma la línea helicoidal de un filo secundario con el eje de la broca. Determina la magnitud del ángulo de ataque en el filo principal.

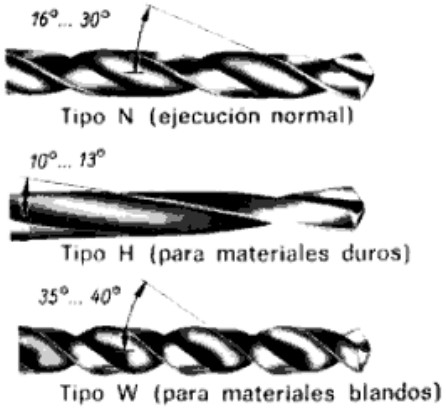

Figura 2.13 Ángulos en la broca de espiral.

2.5.2 Procedimiento de Taladrado mediante CNC

Estudio y aplicación de los ciclos fijos de mecanizado

Los ciclos fijos son las funciones preparatorias que dan mayor potencia a los controles, el permitir realizar la programación de manera sencilla y con gran potencia, permiten arrancar el material en un gran número de pasadas con muy poco código de programación y respetando el perfil programado en todo momento. Algunos de los ciclos fijos se muestran en la Tabla 2.3 [31].

Tabla 2.3 Parámetros recomendados para diferentes algoritmos de ACO.

Ciclo	Descripción
G79	Ciclo fijo definido por el usuario
G80	Anulación de ciclos fijos
G81	Ciclo fijo de taladrado
G82	Ciclo fijo de taladrado con temporización
G83	Ciclo fijo de taladrado profundo

G84	Ciclo fijo de roscado con macho
G85	Ciclo fijo de escariado
G86	Ciclo fijo de mandrinado con retroceso en G00
G87	Ciclo cajera rectangular
G88	Ciclo cajera circular
G89	Ciclo de mandrinado con retroceso en G01

Condiciones en los ciclos

La estructura básica de un ciclo fijo se muestra en la figura 2.14.

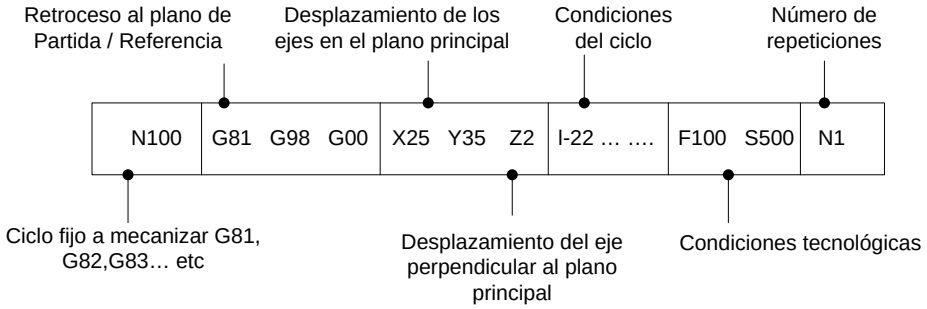

Figura 2.14 Estructura básica de un ciclo fijo.

G81 – Ciclo fijo elegido. Puede ser desde G81 hasta el G89

G98 – Indicación de retroceso al plano de partida o al plano de referencia

XY – Orden de desplazamiento en el plano principal. El control se desplazará hasta el punto indicado de forma paralela al plano activo. No bajará en Z hasta haber alcanzado hasta haber alcanzado dicho punto. Los valores pueden ser absolutos o incrementales. El desplazamiento lo realizará en rápido o a velocidad programada, dependiendo de que esté activa la función G00 o G01.

Z – Desplazamiento en Z hasta la profundidad programada. Esta posición definirá el plano de referencia, también llamado de seguridad, por lo que es importante su definición de forma correcta, no debiendo nunca alcanzar el contacto con la pieza. Los valores pueden ser absolutos o incrementales.

I – En este apartado, se indicarán las diferentes condiciones de ciclo, teniendo en cuenta que no todos los ciclos tendrán las mismas condiciones. En este ejemplo "I" indica profundidad total del ciclo. Los valores pueden ser absolutos o incrementales.

FS – Igual que cualquier otra línea de programación, se podrán indicar todas las condiciones tecnológicas que se crean oportunas.

N – Número de repeticiones que realizará el ciclo. Si se indica valor "0" se detendrá en el punto indicado, pero no realizará el mecanizado. Se pueden programar valores entre 0 y 99, no obstante, si se desean valores superiores se puede programar utilizando un parámetros, con lo que se puede programar hasta 255. Si no se indica "N" al final del ciclo, el control asumirá N1. Como N afecta a la repetición de las coordenadas, si es >1 y no se programa con G91, el control estará siempre en el mismo lugar, repitiendo el mecanizado del ciclo en el mismo sitio.

2.5.3 Ciclo de taladrado

El ciclo de taladrado, como se muestra en la figura 2.15, comienza bajando desde el plano de partida hasta el plano de referencia en velocidad G00 (raya discontinua roja) y a partir de ese punto trabajará en G01 (raya continua azul) hasta alcanzar la longitud total, si se ha programado algún valor en K se detendrá hasta que pase el tiempo solicitado y a

continuación subirá en G00 hasta el plano de referencia o de partida, según se tenga programado G98 o G99 (en la figura sube hasta el plano de partida) [32].

Si se desean realizar más agujeros idénticos, no es necesario programar de nuevo el ciclo, pues desplazándose tan sólo por el plano activo, al alcanzar la posición solicitada realizará de nuevo un taladro y así sucesivamente hasta que se programe un ciclo G80 o cualquier otro ciclo.

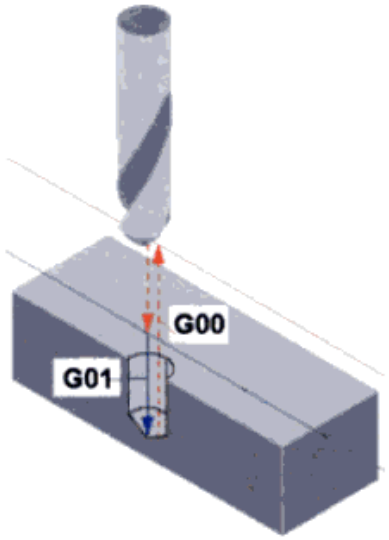

Figura 2.15 Operativa del ciclo G81.

Como un ejemplo [32], se propone un problema el cual nos dice que debemos de mecanizar exclusivamente los taladros. Los taladros son todos de 10 y la broca a usar del mismo diámetro. La longitud de la broca se controla desde la misma punta. El orden en realizar los taladros será el indicado en los números rojos de la figura 2.16.

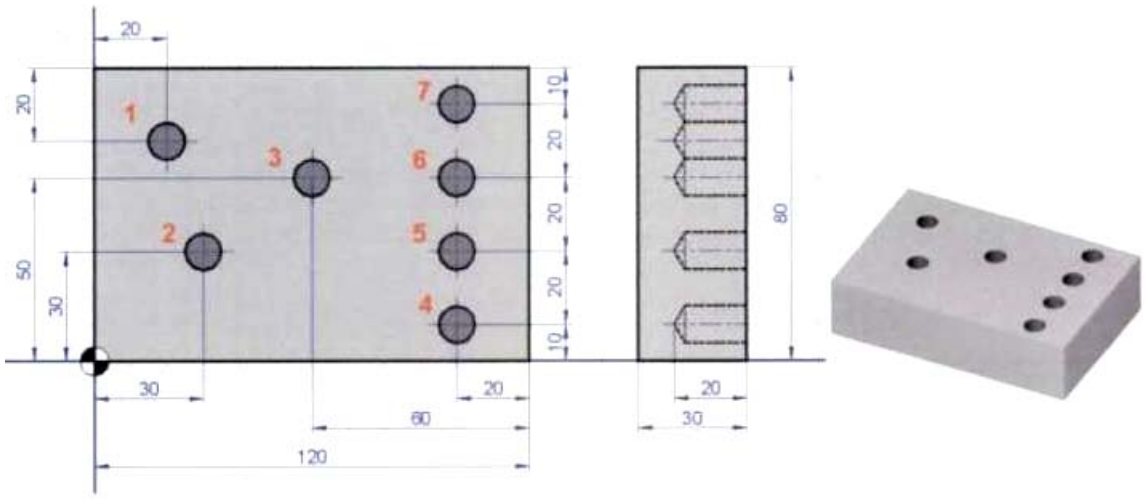

Figura 2.16 Mecanizado de taladros.

La solución se determina mediante la siguiente ejecución de código numérico:

```
N0010 F100 S650 T1.1
N0020 M6
N0030 G0 G90 G43 X-30 Y0 Z50
N0040 G81 G99 X20 Y60 Z2 I-20 M3
N0050 X30 Y30
N0060 X60 Y50
N0070 X100 Y10
N0080 G91 Y20 N3
N0090 G90 G80 X-30 Y0 Z50
N0100 M30
```

- **Línea 0010.** Situación de condiciones tecnológicas iniciales y preparación de herramienta a utilizar.
- **Línea 0020.** Orden de cambio de herramienta 1

- **Línea 0030.** Cabecera y situación de herramienta en un punto conocido y accesible al inicio del mecanizado. Esta situación en Z, define el plano de partida, pues en la línea siguiente se define el ciclo.
- **Línea 0040.** Definición del ciclo de taladrado con G81, orden de regreso a plano de referencia con G99, definición de la situación del taladro no. 1 con la coordenada X20 Y20, definición del plano de referencia con Z2 y se indica la profundidad total del taladro con I-20. La subida posterior al taladrado la realizará al plano de referencia, pues es lo indicado con la función G99.
- **Línea 0050.** Desplazamiento a la coordenada del taladro no. 2 y la realización del taladro en la misma. Como el ciclo del taladrado es modal, es decir está activo, por el simple hecho de desplazar la herramienta a una nueva coordenada realizará el taladro e dicha coordenada.
- **Línea 0060.** Realización del taladro no. 3
- **Línea 0070.** Realización del taladro no. 4
- **Línea 0080.** En esta línea se activa la programación incremental, para poder incrementar los siguientes taladros 5, 6, y 7 en unidades de 20mm mediante la palabra N, que repetirá 3 veces el procedimiento.
- **Línea 0090.** Regresar al plano de referencia con G00 y G80
- **Línea 0100.** Fin del programa

2.6 Descripción de archivos DXF

2.6.1 Intercambio de información entre paquetes de software

Para el intercambio de archivos entre paquetes CAD/CAM es necesario un traductor directo, lo cual se lleva a cabo mediante archivos intermedios, como se muestra en la figura 2.17. Estos archivos pueden presentar alguno de los siguientes estándares de formato: *Drawing Exchange Files (DXF), IGES files y archivos STEP files* [34].

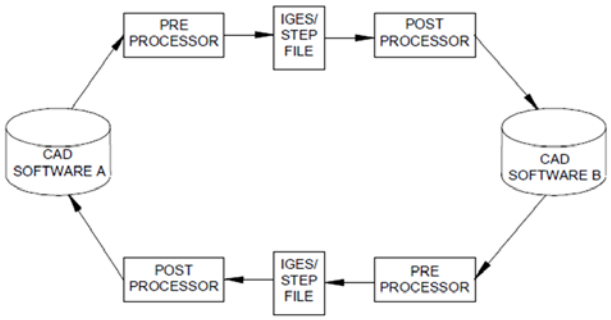

Figura 2.17 Intercambio de datos en CAD utilizando archivos intermedios.

2.6.2 Formato DXF

Un archivo DXF es un tipo de formato de CAD desarrollado por Autodesk® para transferir datos entre AutoCAD® originalmente a cualquier otro programa de CAD o CAM [34]. Actualmente, diversos paquetes comerciales de Diseño Asistido por Computadora (CAD) pueden exportar archivos en formato DXF. Cada una de las ocho secciones en un archivo DXF contiene información relacionada al dibujo: HEADER para información general acerca del dibujo, CLASSES para las clases definidas de la aplicación en las cuales sus instancias aparecen en otras secciones, TABLES para la definición de los

objetos nombrados, BLOCKS para describir las entidades que comprende cada bloque en el dibujo, ENTITIES para las entidades del dibujo, incluyendo referencias a bloques, OBJECTS para los datos aplicados a objetos no-gráficos, usados por AutoLISP y ObjectARX, y THUMBNAILIMAGE para la vista preliminar del archivo DXF [34].

Desde el momento que se comienza a diseñar una pieza en algún paquete CAD, esta misma presenta una ubicación en el espacio virtual dado por coordenadas. El siguiente ejemplo muestra un plano del diseño de una pieza en la figura 2.18 y su correspondiente formato DXF, mismo que se muestra la parte principal del archivo DXF generado en la figura 2.19 (Ver apéndice III).

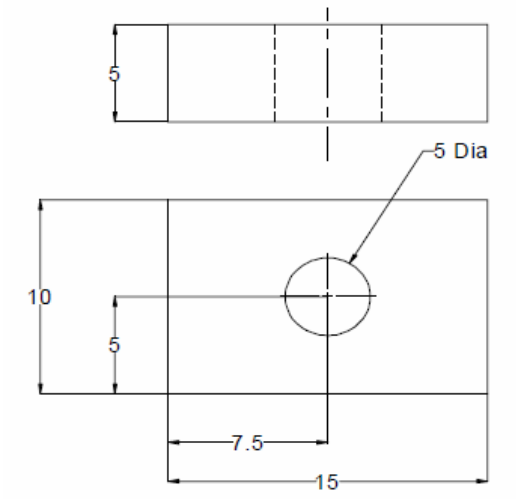

Figura 2.18 Diseño de una pieza.

```
0
SECTION
  2
ENTITIES
  0
LINE
  5
2B
330
1F
100
AcDbEntity
  8
0
100
AcDbLine
 10
137.1906423418079
 20
127.6432581938739
 30
0.0
 11
152.1906423418079
 21
127.6432581938739
 31
0.0
  0
LINE
  5
2C
330
1F
100
AcDbEntity
  8
0
100
AcDbLine
 10
152.1906423418079
 20
127.6432581938739
 30
0.0
 11
152.1906423418079
 21
137.6432581938739
 31
0.0
  0
LINE
  5
2D
330
1F
100
AcDbEntity
  8
0
100
AcDbLine
 10
152.1906423418079
 20
137.6432581938739
 30
0.0
 11
137.1906423418079
 21
137.6432581938739
 31
0.0
  0
LINE
  5
2E
330
1F
100
AcDbEntity
  8

0
100
AcDbLine
10
137.1906423418079
20
137.6432581938739
 30
0.0
 11
137.1906423418079
 21
127.6432581938739
 31
0.0
  0
CIRCLE
  5
31
330
1F
100
AcDbEntity
  8
0
100
AcDbCircle
 10
144.6906423418079
 20
132.6432581938739
 30
0.0
 40
2.5
  0
ENDSEC
```

Figura 2.19 Sección principal correspondiente a la interpretación de la pieza de la Figura 2.18.

Capítulo 3

Optimización de trayectorias para máquinas de control numérico mediante Colonia de Hormigas

3.1 Diseño de la plataforma de software

La plataforma de software diseñada en la presente investigación tiene como objetivo principal la generación de código numérico CNC optimizado mediante Colonia de Hormigas haciendo uso de cómputo de alto rendimiento, además de tener propósitos educativos para el estudio del algoritmo.

3.1.1 Interfaz gráfica

El desarrollo de la plataforma experimental se presenta en el idioma inglés para su registro de autor nacional así como internacional; esta misma se encuentra basada en la tecnología Microsoft® .NET Framework utilizando Visual C#. Debido al diseño de las clases, es posible traducir con facilidad a cualquier otro lenguaje multiplataforma como lo es Java y C++, lenguajes orientados a Objetos.

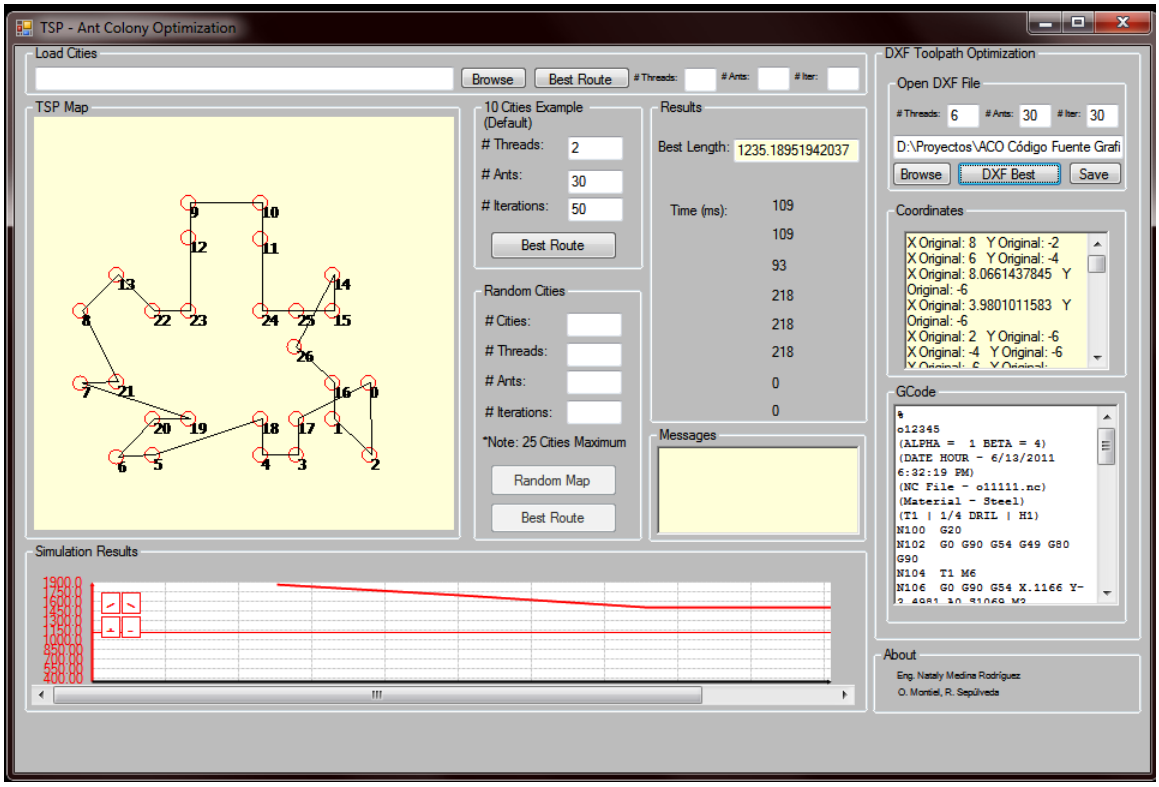

Figura 3.1 Interfaz gráfica de la plataforma experimental.

La plataforma experimental que se muestra en la Figura 3.1 consiste en diversas secciones, la mayoría de ellas con fines educativos para el estudio del algoritmo, y una principal que consiste en la generación del código numérico optimizado, disponible para aplicación en máquinas CNC. A continuación se describen las secciones que comprende la plataforma de software.

A. Vista previa (TSP Map)

En la vista previa de la figura 3.2 se podrán visualizar todos aquellos puntos que representen ciudades (en el caso del estudio del algoritmo para la solución del problema del Agente Viajero) o los orificios que presente una pieza antes diseñada. Se visualizan los puntos y la ruta generada una vez ejecutado el algoritmo de optimización.

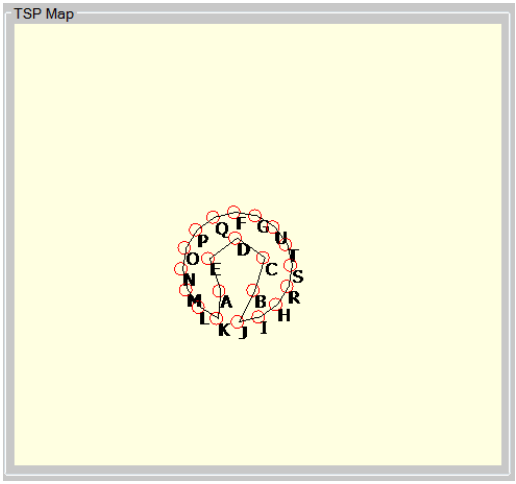

Figura 3.2 Vista previa de un archivo predeterminado de puntos u orificios.

B. Gráfica de ejecución

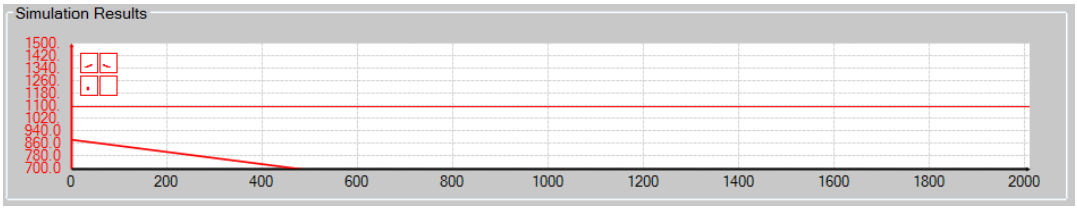

Figura 3.3 Resultados de simulación de manera gráfica.

En la sección de Resultados de Simulación de la figura 3.3 se presenta una gráfica que representa el tiempo de ejecución que le ha tomado al algoritmo para encontrar la distancia óptima del mapa generado.

El eje X representa el tiempo en milisegundos que el algoritmo le ha tomado para encontrar solución al problema, dependiendo de los parámetros iniciales, y el eje Y representa la distancia encontrada.

En esta gráfica de la Figura 3.3 se puede visualizar cómo ha evolucionado el algoritmo para encontrar un óptimo, lo cual nos da una idea de la convergencia del mismo medida en unidades de tiempo.

C. Resultados y mensajes

Figura 3.4 Resultados numéricos y mensajes.

El modelo de programación se encuentra basado en cómputo paralelo, utilizando procesadores multinúcleo; es posible visualizar en el área de resultados numéricos el tiempo que le ha tomado a cada núcleo (virtual o físico) completar las tareas de manera sincronizada; además se puede visualizar la distancia final encontrada como la solución óptima. En el área de mensajes aparecerá cualquier advertencia, error o asignación durante la ejecución del algoritmo como se muestra en la figura 3.4.

D. Mapa predeterminado y generación de mapas aleatorios

Figura 3.5 Mapa predeterminado y generación de mapas aleatorios.

En esta sección mostrada en la figura 3.5, se presenta un mapa predeterminado de 10 nodos para ejecutar el algoritmo y visualizar su comportamiento de manera inicial. En la sección de generación de mapas aleatorios, podemos generar un máximo de 25 nodos de manera aleatoria; el control del botón de ejecución y generación se desactiva cuando otra función del programa es activada.

E. Interpretación de mapas por medio de archivos XML

Figura 3.6 Interpretación de mapas en archivos XML.

El programa se encuentra diseñado para que pueda interpretar mapas diseñados en archivos XML, con el propósito presentar una plataforma de software que permita la

interacción con formatos estándar, mostrando esta sección como se muestra en la figura 3.6.

F. Optimización de Trayectorias para Máquinas de Control Numérico

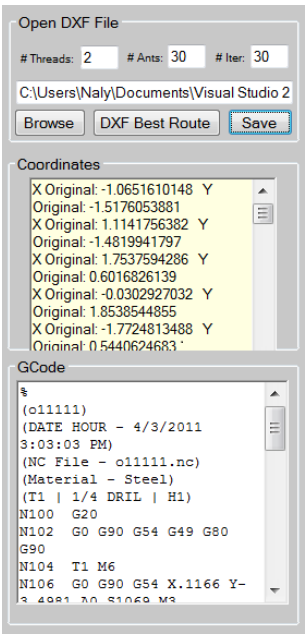

Figura 3.7 Sección de interpretación de archivo DXF y la generación de Código Numérico.

Esta sección como se muestra en la figura 3.7 permite realizar una búsqueda de algún archivo DXF deseado que contenga el dibujo de una pieza y orificios en la misma para ser procesados por el algoritmo de Optimización por Colonia de Hormigas para así encontrar la trayectoria óptima en distancia. En seguida se procede a generar el código numérico relacionado a la pieza que será maquinada, en este caso taladrada. En esta parte se puede guardar un archivo directamente en una unidad de disco para ser ejecutado en la máquina CNC directamente.

3.1.2 Clases principales

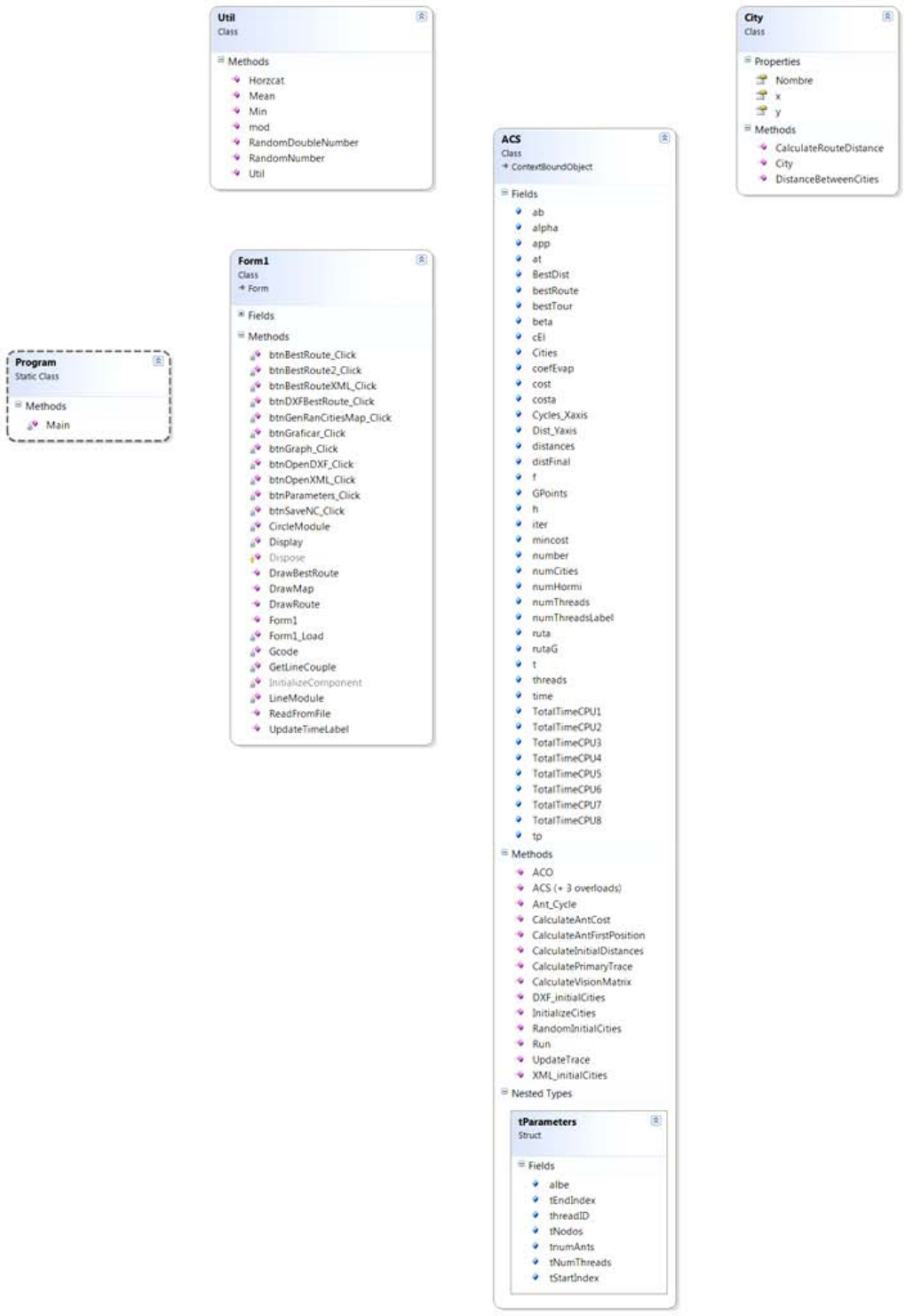

Figura 3.8 Diagrama general de las clases de la plataforma de software.

En la figura 3.8 se muestra el diagrama de clases de la plataforma de software, misma que contiene el algoritmo por Colonia de Hormigas. La clase principal denotada como ACS contiene los métodos principales necesarios para la generación de todo tipo de trayectorias optimizadas por medio de distancias Euclidianas. Los métodos ACS que forman parte del algoritmo son:

- *Ant_Cycle y ACO.* Estas funciones son encargadas de procesar todos aquellos parámetros de inicialización y ejecución de las funciones heurísticas.
- *CalculateAntFirstPosition*. Inicializando el algoritmo con hormigas en cada nodo de manera aleatoria hacen que la búsqueda sea mucho más rápida que inicializando las hormigas en un nodo en específico.
- *CalculateAntCost*. Cálculo del costo que lleva la ruta generada por cada iteración o búsqueda de la hormiga, para efectos de evaluación de la misma y selección de la mejor ruta con menor costo.
- *CalculateInitialDistances.* El cálculo de las distancias se encuentra directamente relacionado al cálculo del costo, ya que el costo para este algoritmo es la distancia euclidiana entre nodos o ciudades del TSP.
- *CalculatePrimaryTrace.* El trazado primario consiste en las trayectorias con la concentración de feromona inicial, el mejor caso es inicializar dicho parámetro con un valor de cero.
- *CalculateVisionMatrix.* La matriz de visión contiene el valor heurístico, definido en el capítulo 2 como $\eta = \frac{1}{d_{ij}}$, esto es, el recíproco de la distancia entre dos nodos o vértices.
- *InitializeCities.* Esta función se le ha nombrado con el término de ciudades debido a que la solución del algoritmo es el cálculo de la ruta óptima para un Agente

Viajero. En esta función principalmente se inicializan todos aquellos nodos que representan una ciudad, o en el caso de la aplicación a maquinado, representan un orificio en particular, dado por un nombre, y coordenada en un plano 2D.

- *UpdateTrace.* La actualización del rastro de feromona se encuentra dada por $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \ \forall(i,j) \in L$, donde ρ es la tasa de evaporación descrita en el capítulo 2. Es la función que cicla por cada hormiga y por cada nodo.

La clase `City` contiene un constructor que recibe un nombre, y una coordenada en X y Y para cada nodo agregado a la lista del mapa. Las funciones que se realizan principalmente en esta clase son el cálculo de la distancia entre nodos o ciudades, y el cálculo de la distancia total de la ruta una vez que esta misma haya sido evaluada como la mejor por el algoritmo ACS.

La clase `Util` contiene las funciones principales para el cálculo de números aleatorios con base a una misma semilla, cálculo de mínimos y máximos, la concatenación de matrices, entre otras herramientas matemáticas utilizadas dentro del algoritmo.

Finalmente la clase `Form1` contiene todas las cuestiones gráficas y generación de mapas por diferentes métodos, ya sea de manera aleatoria, por archivos XML o generados mediante archivos DXF. Esta clase contiene también la función dedicada a la generación de código G, pues es la clase que representa la aplicación en general. Por último, la clase `Program` es la clase principal, la cual contiene el método *main* o principal que manda a llamar a la Forma para su ejecución.

3.1.3 Procesamiento en paralelo del algoritmo

El algoritmo ACS comprende el siguiente procedimiento:

Algoritmo 3.1 Algoritmo clásico de ACO.

Entrada: m: número de hormigas
n: número de nodos
α: parámetro correspondiente al rastro
β: parámetro correspondiente a la información heurística
Dato: i: contador de iteraciones
P_{ij}: matriz de probabilidades
τ_{ij}: rastro de feromona
η_{ij}: matriz de heurística $1/d_{ij}$
Salida: S: mejor solución

1. **comenzar**
2. inicializar() //coeficientes de evaporación, matrices de distancias
3. **for** c1 = 1 **to** i
   for c2 = 1 **to** m
   for c3 = 1 **to** n
   S ← construirSolución(c3)
   $$P_{ij}(c3) = \begin{cases} \frac{\tau_{ij}^{\alpha}\,\eta_{ij}^{\beta}}{\sum \tau_{ij}^{\alpha}\,\eta_{ij}^{\beta}} \\ 0 \end{cases}$$
   end for
   end for
4. Guardar mejor solución en S
5. Actualizar rastro τ_{ij}
6. **end for**
7. **regresar** S
8. **terminar**

En el proceso de construcción, cada hormiga es inicialmente asignada de manera aleatoria a un nodo inicial. En cada paso de la iteración, un nodo no visitado se agrega a una ruta parcial, por lo que la construcción de la solución terminará una vez que todos los nodos o ciudades sean visitados.

A. Metodología propuesta

En la figura 3.9 se puede apreciar la división de tareas para efectuar procedimientos utilizando paralelismo.

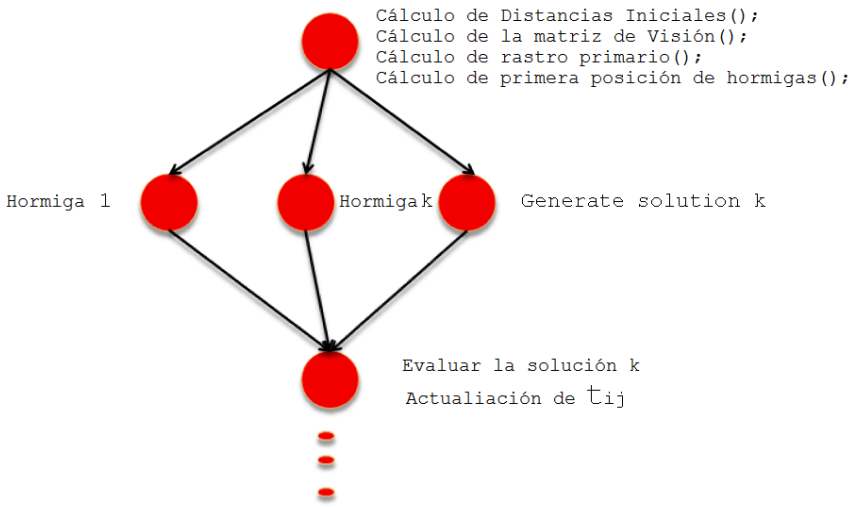

Figura 3.9 Implementación en paralelo en la generación de las soluciones.

Al comienzo del algoritmo, un proceso maestro inicializa la información y genera k procesos (uno por cada hormiga) y transmite a su vez la información. Al comienzo del ciclo, la matriz τ_{ij} (rastro de feromona) es enviada a cada proceso y los cálculos (generación de las soluciones) se realizan en paralelo. Entonces, las soluciones y sus evaluaciones vuelven al proceso maestro para la actualización de todos aquellos

parámetros involucrados, entonces la matriz τ_{ij} es actualizada y un nuevo ciclo comienza con estos nuevos parámetros.

B. Sincronización con la actualización de la matriz τ_{ij}

Al final de cada ciclo, la matriz τ_{ij} tiene que ser actualizada por las hormigas en el siguiente ciclo, independientemente de que en nuestra implementación no existe la necesidad de mostrar explícitamente la comunicación antes del procedimiento de actualización; la hebra de ejecución maestra tiene que esperar hasta que todos los procesos (hormigas) del ciclo actual termine de realizar los cálculos y evaluar sus secuencias de trabajo, por lo que esta barrera de sincronización es independiente del modelo utilizado y no se puede evitar sin alterar el método original.

3.2 Experimentos

Para analizar la eficiencia del algoritmo ha sido conveniente diseñar algunos experimentos que consisten en prototipos de CAD que representan en 3D placas con una serie de perforaciones, mismas que deberán ser maquinadas una vez generado el código numérico.

El diseño consiste en una placa de $10" \times 12" \times 0.25"$ con n perforaciones de 1/4" de diámetro por orificio. El experimento deberá ejecutarse por lo menos 30 veces para efectuar los cálculos correspondientes. A continuación se presentan los experimentos diseñados para el análisis de resultados, presentando a los mismos en el capítulo 4.

A. EXPERIMENTO 1

En la tabla 3.1 se muestra el primer experimento de un diseño de placa con perforaciones mostrada en la figura 3.10. Este primer experimento consiste en analizar el algoritmo, variando los parámetros de control α y β.

Tabla 3.1 Parámetros del experimento 1.

Experimento 1 – A	**Experimento 1 – B**	**Experimento 1 – C**
Número de orificios: 10	Número de orificios: 10	Número de orificios: 10
Número de hormigas: 30	Número de hormigas: 30	Número de hormigas: 30
Número de iteraciones: 30	Número de iteraciones: 30	Número de iteraciones: 30
Alfa: 1	Alfa: 2	Alfa: 2.5
Beta: 4	Beta: 4.5	Beta: 4

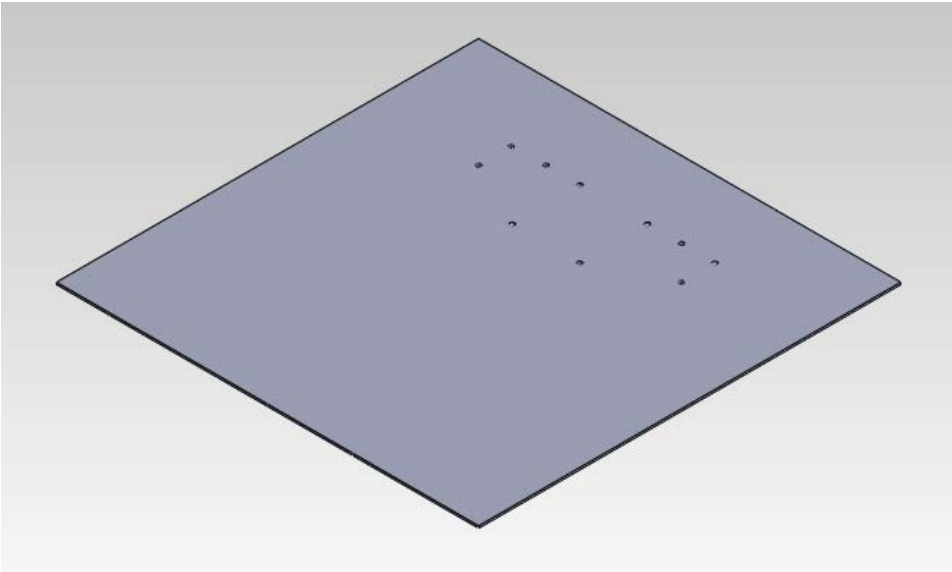

Figura 3.10 Experimento 1 con 10 orificios.

B. EXPERIMENTO 2

En la tabla 3.2 se muestra el segundo experimento de un diseño de placa con perforaciones mostrada en la figura 3.11. Este segundo experimento consiste en analizar el algoritmo, variando los parámetros de control α y β.

Tabla 3.2 Parámetros del experimento 2.

Experimento 2 – A	**Experimento 2 – B**	**Experimento 2 – C**
Número de orificios: 27	Número de orificios: 27	Número de orificios: 27
Número de hormigas: 30	Número de hormigas: 30	Número de hormigas: 30
Número de iteraciones: 30	Número de iteraciones: 30	Número de iteraciones: 30
Alfa: 1	Alfa: 2	Alfa: 2.5
Beta: 4	Beta: 4.5	Beta: 4

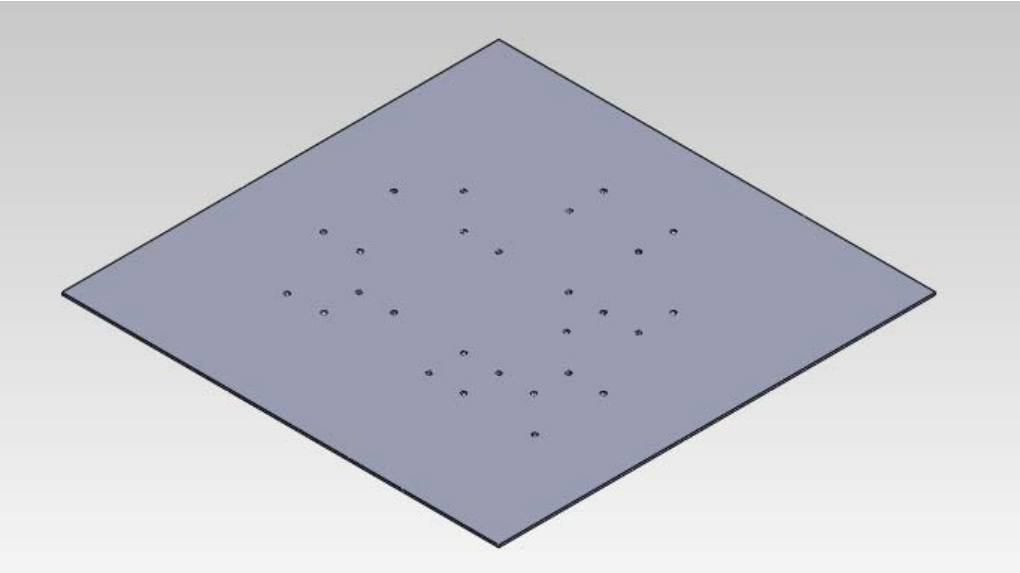

Figura 3.11 Experimento 2 con 27 orificios.

C. EXPERIMENTO 3

En la tabla 3.3 se muestra el tercer experimento de un diseño de placa con perforaciones mostrada en la figura 3.12. Este tercer experimento consiste en analizar el algoritmo, variando los parámetros de control α y β.

Tabla 3.3 Parámetros de experimento 3.

Experimento 3 – A	**Experimento 3 – B**	**Experimento 3 – C**
Número de orificios: 45	Número de orificios: 45	Número de orificios: 45
Número de hormigas: 70	Número de hormigas: 70	Número de hormigas: 70
Número de iteraciones: 70	Número de iteraciones: 70	Número de iteraciones: 70
Alfa: 1	Alfa: 2	Alfa: 2.5
Beta: 4	Beta: 4.5	Beta: 4

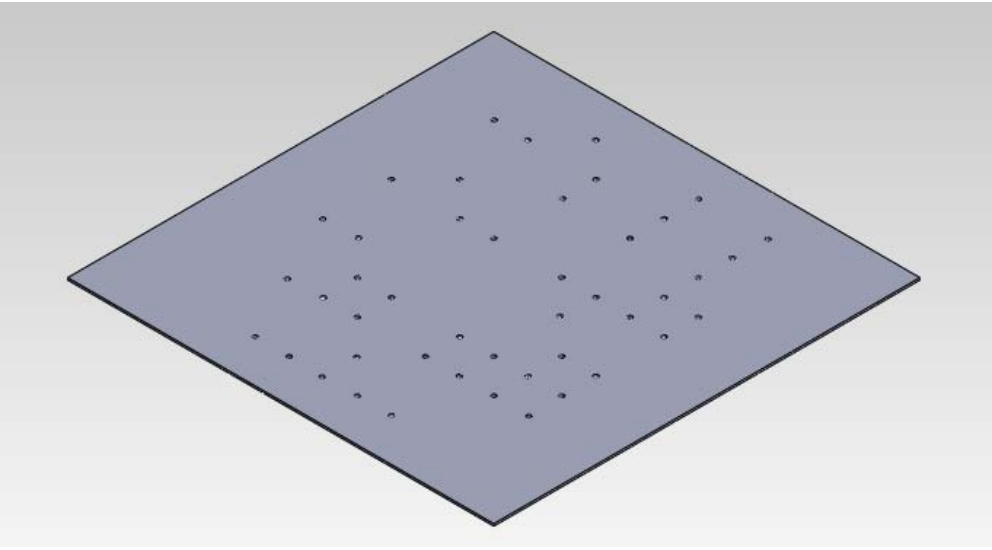

Figura 3.12 Experimento 3 con 45 orificios.

D. EXPERIMENTO 4

En la tabla 3.4 se muestra el cuarto experimento de un diseño de placa con perforaciones mostrada en la figura 3.13. Este cuarto experimento consiste en analizar el algoritmo, variando los parámetros de control α y β.

Tabla 3.4 Parámetros de experimento 4.

Experimento 4 – A	**Experimento 4 – B**	**Experimento 4 – C**
Número de orificios: 65	Número de orificios: 65	Número de orificios: 65
Número de hormigas: 100	Número de hormigas: 100	Número de hormigas: 100
Número de iteraciones: 100	Número de iteraciones: 100	Número de iteraciones: 100
Alfa: 1	Alfa: 2	Alfa: 2.5
Beta: 4	Beta: 4.5	Beta: 4

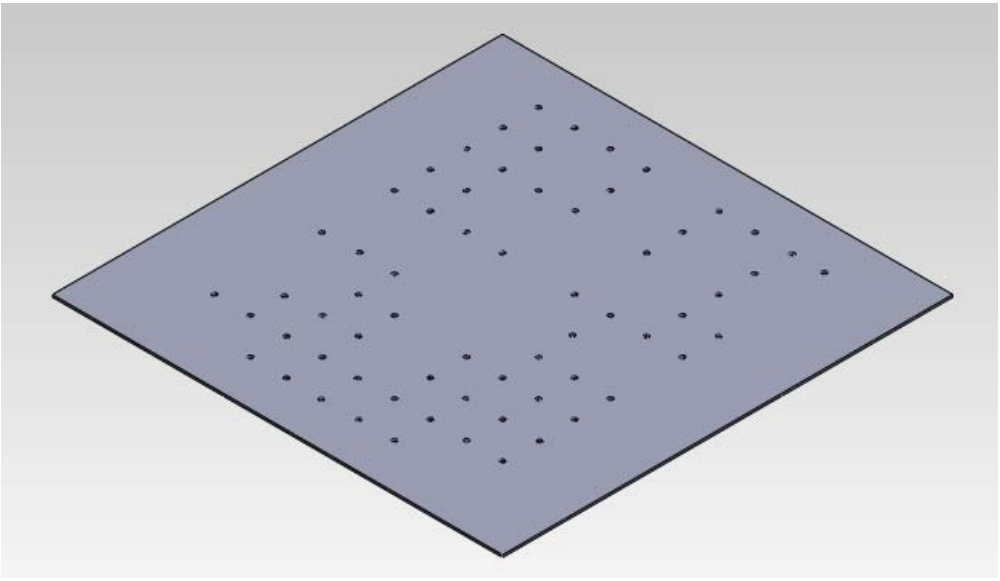

Figura 3.13 Experimento 4 con 65 orificios.

E. EXPERIMENTO 5

En la tabla 3.5 se muestra el quinto experimento de un diseño de placa con perforaciones mostrada en la figura 3.14. Este quinto experimento consiste en analizar el algoritmo, variando los parámetros de control α y β.

Tabla 3.5 Parámetros de experimento 5.

Experimento 5 – A	**Experimento 5 – B**	**Experimento 5 – C**
Número de orificios: 70	Número de orificios: 70	Número de orificios: 70
Número de hormigas: 125	Número de hormigas: 125	Número de hormigas: 125
Número de iteraciones: 125	Número de iteraciones: 125	Número de iteraciones: 125
Alfa: 1	Alfa: 2	Alfa: 2.5
Beta: 4	Beta: 4.5	Beta: 4

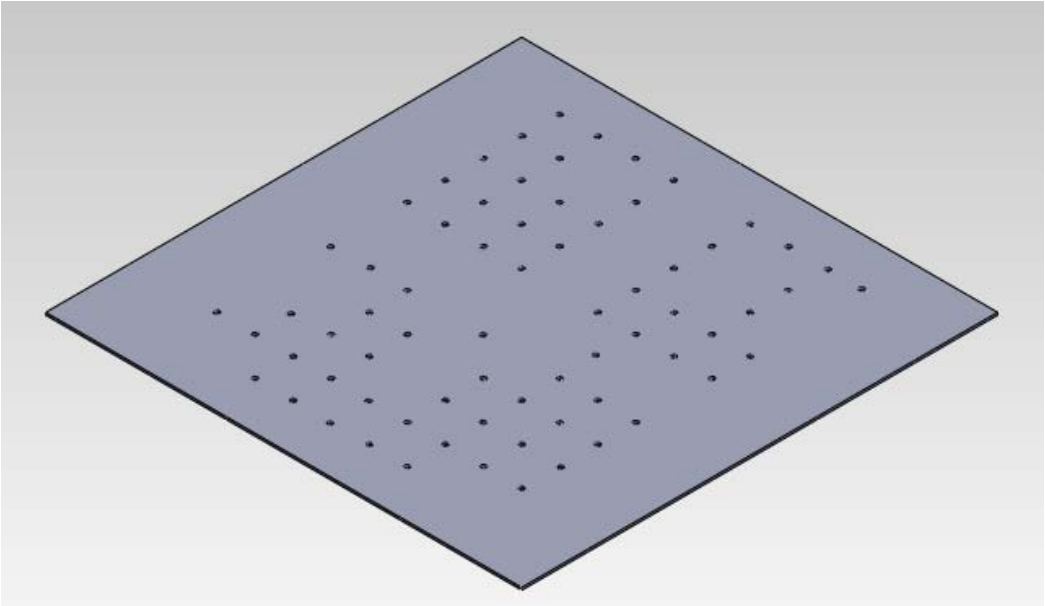

Figura 3.14 Experimento 5 con 70 orificios.

3.3 Interpretación de Archivos DXF

Para la interpretación del archivo DXF, se diseñó una pieza de prueba mediante el uso del paquete comercial de CAD SolidWorks® como se muestra en la figura 3.15.

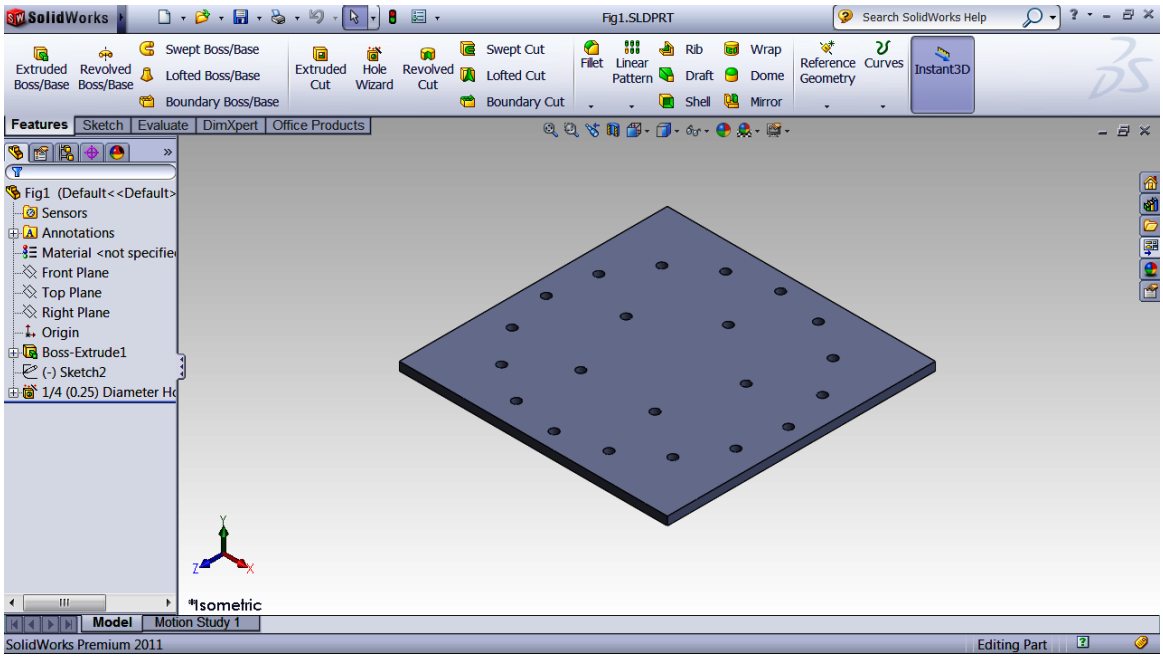

Figura 3.15 Diseño de una pieza de prueba utilizando SolidWorks®.

El diseño de la primera pieza de prueba consistió en una placa simétrica con 21 orificios ordenados de manera circular; el propósito de la geometría regular fue la facilitación de la interpretación de un archivo DXF mismo que se genera en el programa de CAD, como se muestra en la figura 3.16.

Cabe destacar que para la conversión de un archivo CAD a un archivo en formato DXF se debe especificar la vista, en donde esta misma debe de contener la mayor información posible para efectos de maquinado de la misma.

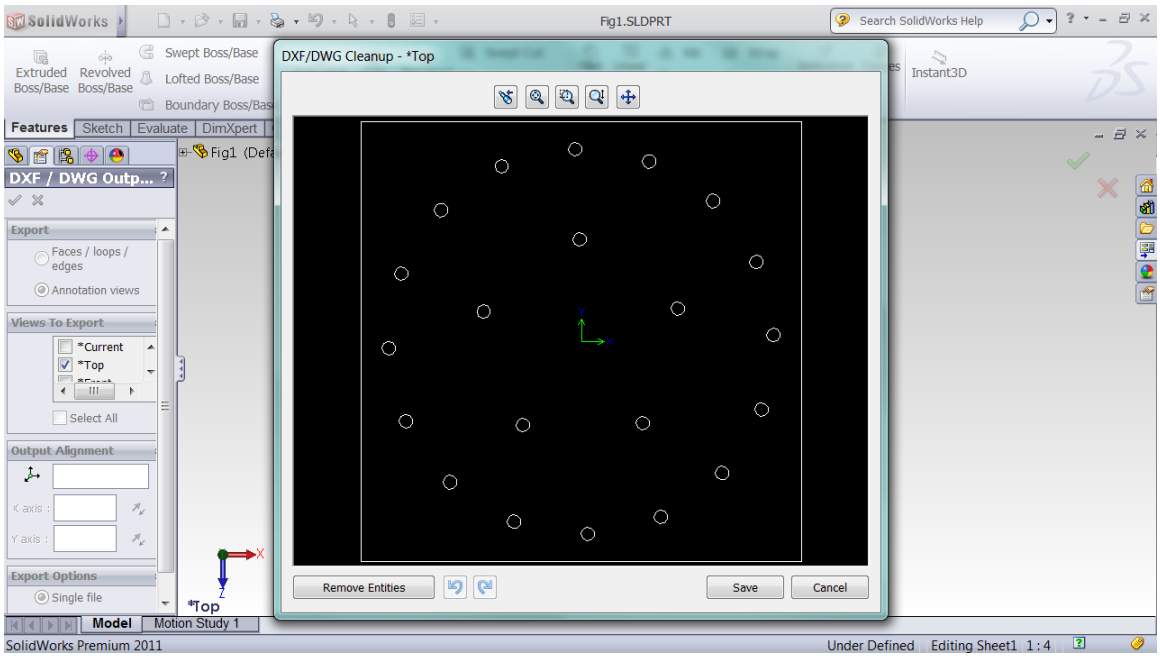

Figura 3.16 Preparación del archivo DXF en SolidWorks®.

El siguiente código de la figura 3.17 muestra una sección del archivo DXF para un círculo (orificio en el caso de taladrado en la pieza), mostrando las coordenadas y el diámetro del mismo.

```
  0
SECTION
  2
ENTITIES
  0
CIRCLE
  5
48
330
1F
100
AcDbEntity
  8
0
  6
Continuous
 62
     7
370
    15
100
AcDbCircle
 10
-1.0651610148
 20
-1.5176053881
 30
0.0
 40
0.1249999999999998
```

Figura 3.17 Sección de un documento DXF que muestra las coordenadas y dimensiones de un orificio.

La ubicación de este orificio se encuentra dado por las coordenadas `-1.06516` y `-1.5176` con respecto al origen en 0,0. La interpretación de un archivo en formato DXF ha seguido el siguiente procedimiento en la presente investigación:

```
ReadFromFile(file.dxf)

While(line2 ¡= “EOF”)
     If line1 = 0 and line2 = “LINE”
          LineModule(reader)
     Else if line1 = 0 and line2 = “CIRCLE”
          CircleModule(reader)
     ...
     GetLineCouple
```

Una vez recuperado los datos que contienen la información de la ubicación en la pieza, se tienen dos opciones para el procesamiento; la primera de ellas consta del procedimiento para la visualización de los orificios en la interfaz gráfica diseñada y la segunda es la optimización del código G81 que se presenta en la sección 3.3 por medio de la obtención de las coordenadas correspondientes de cada orificio de la pieza.

Para la visualización de los orificios en la interfaz gráfica se debe considerar que el formato DXF trabaja con el sistema de coordenadas WCS (World Coordinate System) que es el sistema más utilizado en dibujo técnico y en agencias espaciales, como la NASA para la ubicación de objetos [35].

Como los gráficos computacionales utilizan un Sistema de Coordenadas de Pixeles (Pixel Coordinate System - PCS) es necesario una transformación de WCS

[36] a PCS [37] normalizando los datos obtenidos de la interpretación del archivo DXF. El procedimiento para la normalización de los datos WCS para el presente trabajo ha sido diseñado con base en el siguiente procedimiento.

```
For each data in WCS values
     Get min x, y (negative values)
End for
     Sum abs(x) + x_i in WCS, Sum abs(y) + y_i in WCS
```

Esto hace posible la visualización de los orificios en el plano basado en el Sistema de Coordenadas de Pixeles [37] [38], en donde todos los valores son positivos, como se muestra en la figura 3.18.

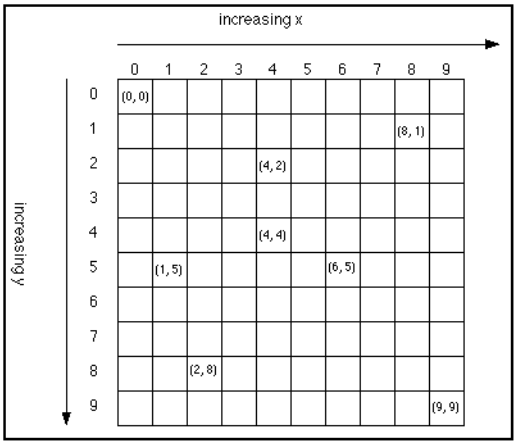

Figura 3.18 Sistema de Coordenadas de Pixeles.

3.4 Optimización de código G81 para Taladrado

La metodología del presente trabajo se encuentra basada en un procedimiento riguroso desde la interpretación del archivo DXF generado por algún paquete comercial de CAD, la obtención de la información válida para así optimizar la trayectoria que seguirá la herramienta de taladrado por medio de una Metaheurística conocida como Optimización por Colonia de Hormigas para así generar el código numérico que será ejecutado en una máquina de control numérico; dicho procedimiento se puede visualizar en la figura 3.19.

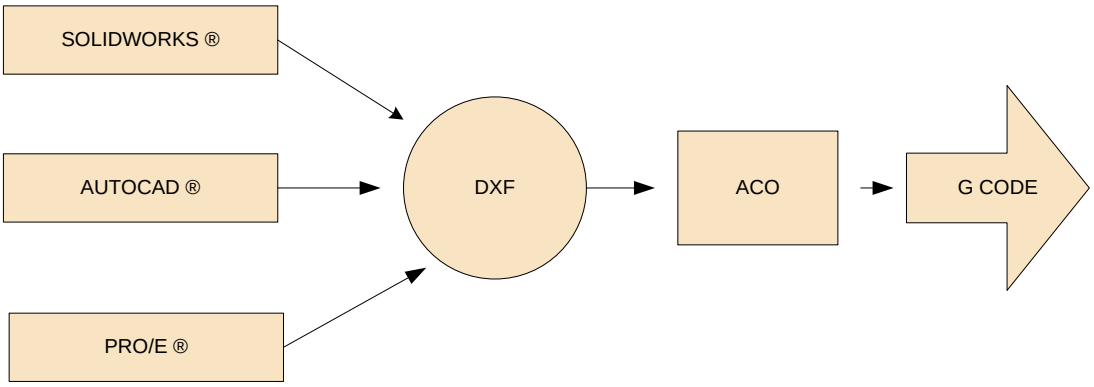

Figura 3.19 Interpretación de archives DXF y optimización de trayectoria de maquinado.

La primera fase consiste en el diseño en CAD de alguna pieza que presente una serie de perforaciones a realizar; y como primera versión del software desarrollado, las piezas a diseñar en CAD deberán ser planas, es decir, cualquier placa que presente una serie de orificios, como lo son las tarjetas de circuitos impresos.

La segunda fase consiste en la generación del archivo DXF, que es el formato estándar entre diversos paquetes CAD/CAM, mismo que será interpretado

en la fase tres, obteniendo información válida requerida por el algoritmo de optimización en la fase cuatro; la información es válida cuando esta describa la ubicación exacta de los orificios de la pieza.

Cuando se encuentra en la fase cuatro, la información obtenida de la interpretación del archivo DXF ingresa al algoritmo de optimización por Colonia de Hormigas, dando como resultado la mejor ruta basada en la distancia más corta; esto producirá una trayectoria para la herramienta de taladrado, ahorrando tiempos de maquinado.

La manipulación de la herramienta de taladrado se hace a partir de la fase cinco, que es la generación del código numérico utilizado en la mayoría de las máquinas CNC. Finalizando el proceso con la ejecución de este programa numérico en la máquina CNC, continuando con el proceso de manufactura.

Capítulo 4

Resultados experimentales

Para el desarrollo del presente trabajo de investigación se ha diseñado la plataforma de software experimental basada en la tecnología Microsoft® .NET Framework, utilizando una computadora con las siguientes especificaciones técnicas:

- Procesador Intel® Core™ i7 CPU 930 @ 2.80 GHz
- Memoria RAM: 6 GB
- Tipo de sistema: 64 bits

4.1 Análisis de eficiencia del algoritmo

Para describir el análisis correspondiente del algoritmo por Colonia de Hormigas mediante el uso de cómputo de alto rendimiento, se procede a ejecutar los experimentos mencionados en el capítulo 3. Para la descripción del análisis, se ejecutó el algoritmo 30 veces por cada experimento para obtener una muestra promedio.

A. Experimento 1

La figura 4.1 muestra el primer experimento para las pruebas del algoritmo basado en cómputo paralelo, generando la trayectoria optimizada de la herramienta de taladrado como se muestra en el mapa de resultados visuales. Las tablas 4.1, 4.2 y 4.3 muestran los resultados experimentales correspondientes relacionados al tiempo de ejecución del algoritmo de acuerdo al número de procesadores utilizados; además se determina la ganancia de velocidad y la eficiencia del algoritmo.

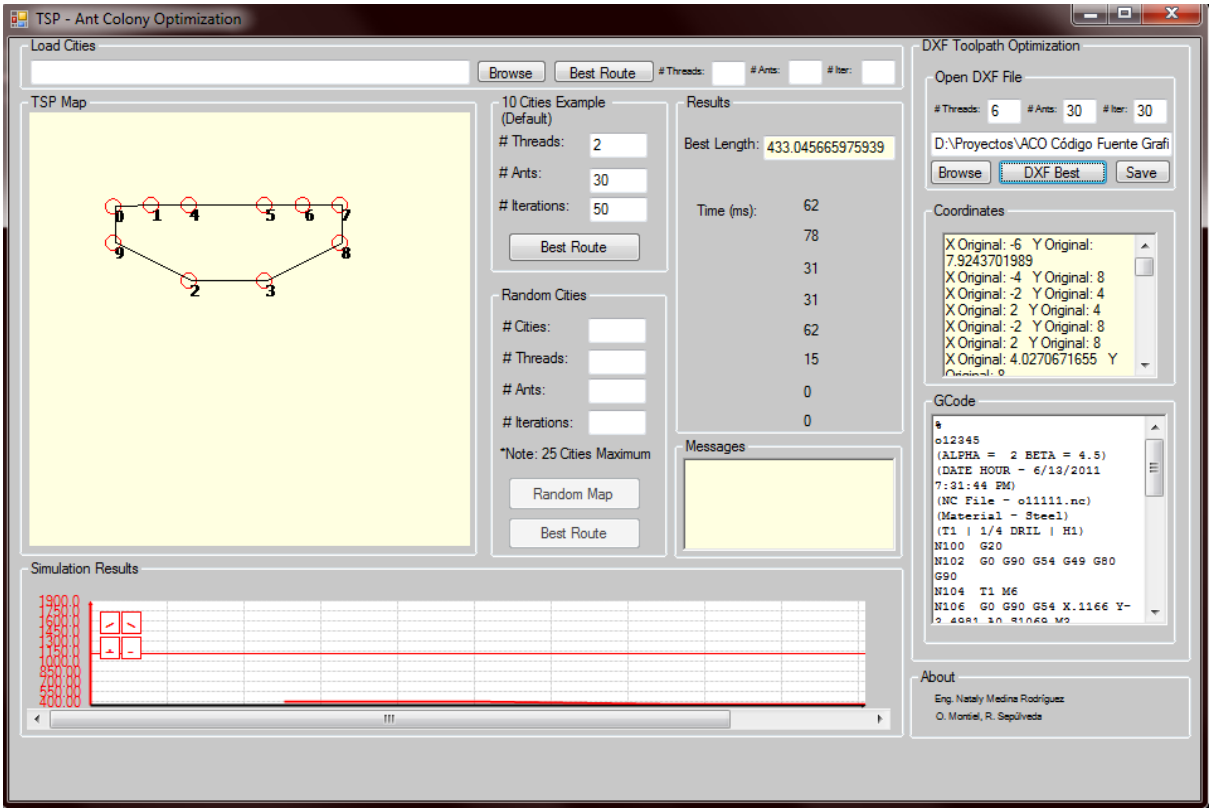

Figura 4.1 Resultado gráfico efectuado por el experimento 1.

Tabla 4.1 Resultados experimento 1 – A.

10 nodos, 30 hormigas y 30 iteraciones, alpha 1, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	78	-	-
2	40	1.95	0.975
3	38	2.052632	0.684210526
4	20	3.9	0.975
5	19	4.105263	0.821052632
6	15	5.2	0.866666667

Tabla 4.2 Resultados experimento 1 – B.

10 nodos, 30 hormigas y 30 iteraciones alpha 2, beta 4.5			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	93	-	-
2	60	1.55	0.775
3	50	1.86	0.62
4	30	3.1	0.775
5	20	4.65	0.93
6	17	5.4705882	0.91176471

Tabla 4.3 Resultados experimento 1 – C.

10 nodos, 30 hormigas y 30 iteraciones alpha 2.5, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	92	-	-
2	70	1.314286	0.657142857
3	64	1.4375	0.479166667
4	25	3.68	0.92
5	19	4.842105	0.968421053
6	17	5.411765	0.901960784

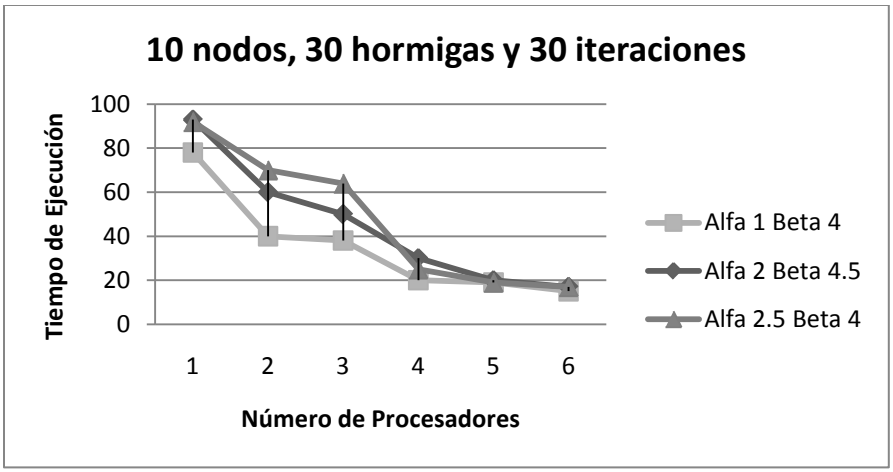

Figura 4.2 Tiempos de ejecución de acuerdo al número de procesadores para el experimento 1.

En la figura 4.2 se puede observar la disminución del tiempo de ejecución entre más procesadores se utilicen.

B. Experimento 2

La figura 4.3 muestra el segundo experimento para las pruebas del algoritmo basado en cómputo paralelo, generando la trayectoria optimizada de la herramienta de taladrado como se muestra en el mapa de resultados visuales. Las tablas 4.4, 4.5 y 4.6 muestran los resultados experimentales correspondientes relacionados al tiempo de ejecución del algoritmo de acuerdo al número de procesadores utilizados; además se determina la ganancia de velocidad y la eficiencia del algoritmo.

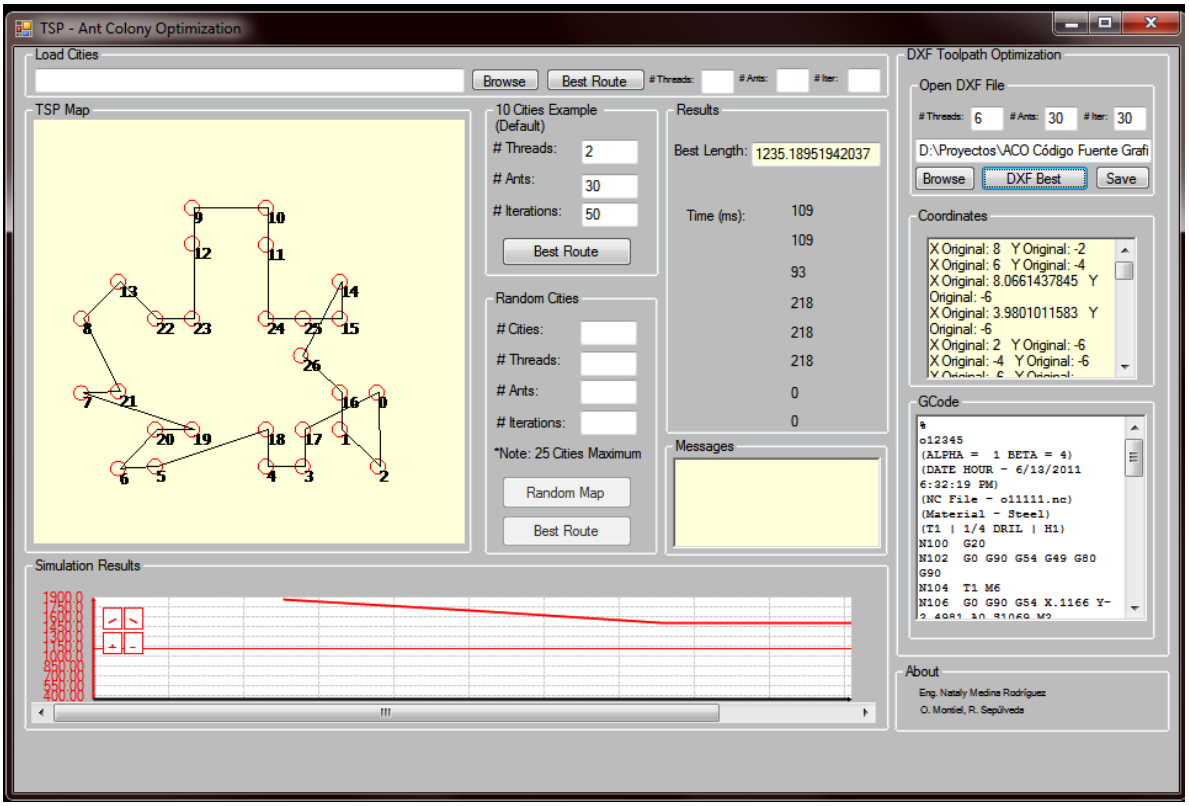

Figura 4.3 Resultado gráfico efectuado por el experimento 2.

Tabla 4.4 Resultados experimento 2 – A.

27 nodos, 30 hormigas y 30 iteraciones, alpha 1, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	686	-	-
2	500	1.372	0.686
3	453	1.514349	0.504782929
4	225	3.048889	0.762222222
5	200	3.43	0.686
6	153	4.48366	0.747276688

Tabla 4.5 Resultados experimento 2 – B.

27 nodos, 30 hormigas y 30 iteraciones, alpha 2, beta 4.5			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	748	-	-
2	523	1.4302103	0.71510516
3	419	1.7852029	0.59506762
4	225	3.3244444	0.83111111
5	156	4.7948718	0.95897436
6	134	5.5820896	0.93034826

Tabla 4.6 Resultados experimento 2 – C.

27 nodos, 30 hormigas y 30 iteraciones, alpha 2.5, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	790	-	-
2	523	1.510516	0.755258126
3	419	1.885442	0.628480509
4	225	3.511111	0.877777778
5	178	4.438202	0.887640449
6	134	5.895522	0.982587065

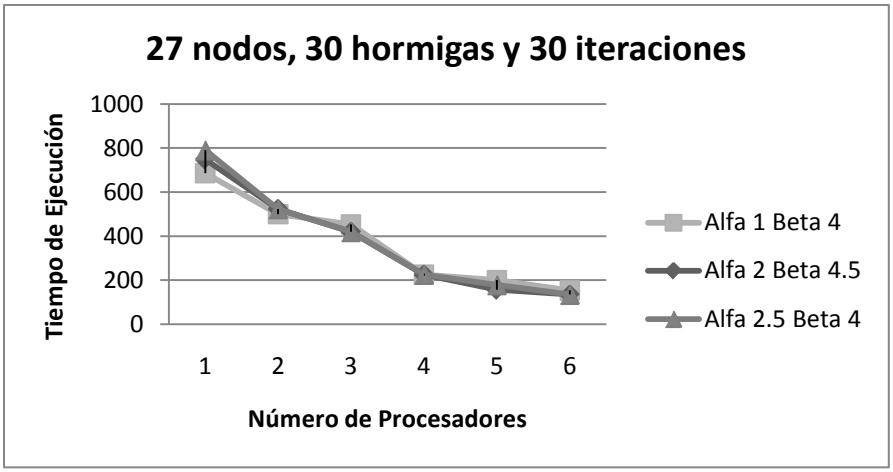

Figura 4.4 Tiempos de ejecución de acuerdo al número de procesadores para el experimento 2.

En la figura 4.4 se puede observar que en este experimento también se produce una disminución de los tiempos de ejecución del algoritmo entre más procesadores se utilicen.

C. Experimento 3

La figura 4.5 muestra el tercer experimento para las pruebas del algoritmo basado en cómputo paralelo, generando la trayectoria optimizada de la herramienta de taladrado como se muestra en el mapa de resultados visuales. Las tablas 4.7, 4.8 y 4.9 muestran los resultados experimentales correspondientes relacionados al tiempo de ejecución del algoritmo de acuerdo al número de procesadores utilizados; además se determina la ganancia de velocidad y la eficiencia del algoritmo.

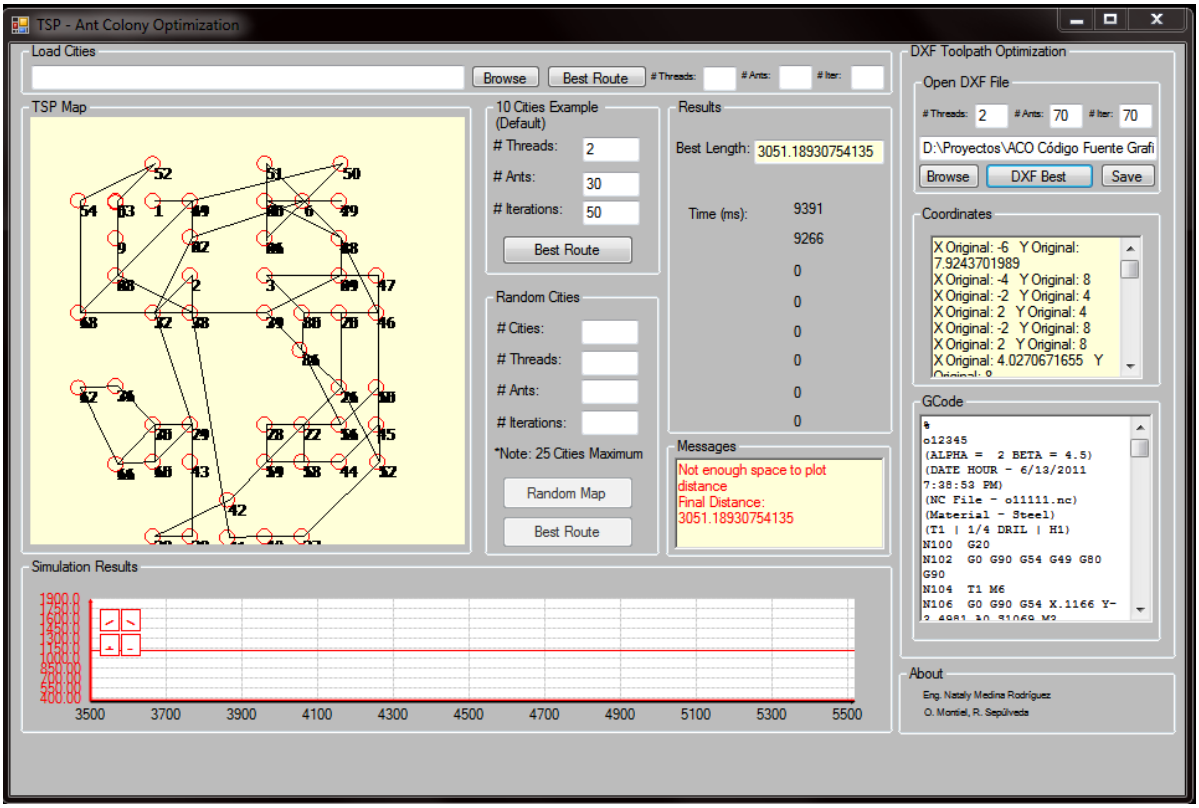

Figura 4.5 Resultado gráfico efectuado por el experimento 3.

Tabla 4.7 Resultados experimento 3 – A.

45 nodos, 70 hormigas y 70 iteraciones, alpha 1, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	13088	-	-
2	12058	1.08542	0.542710234
3	9857	1.327787	0.442595786
4	7058	1.85435	0.463587419
5	3025	4.326612	0.865322314
6	3014	4.342402	0.723733687

Tabla 4.8 Resultados experimento 3 – B.

45 nodos, 70 hormigas y 70 iteraciones, alpha 2, beta 4.5			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	18174	-	-
2	12058	1.5072151	0.75360756
3	9857	1.8437659	0.61458862
4	7058	2.5749504	0.6437376
5	5000	3.6348	0.72696
6	4983	3.6472005	0.60786675

Tabla 4.9 Resultados experimento 3 – C.

45 nodos, 70 hormigas y 70 iteraciones, alpha 2.5, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	18033	-	-
2	12058	1.495522	0.747760823
3	8798	2.04967	0.68322346
4	5890	3.06163	0.76540747
5	5000	3.6066	0.72132
6	3635	4.960935	0.826822558

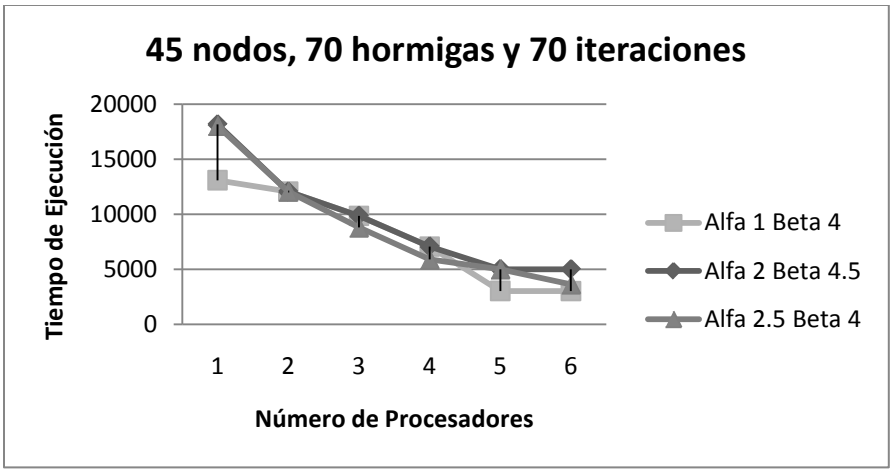

Figura 4.6 Tiempos de ejecución de acuerdo al número de procesadores para el experimento 3.

En la figura 4.6 se puede observar que de forma similar, este experimento presenta una reducción de tiempos de ejecución del algoritmo.

D. Experimento 4

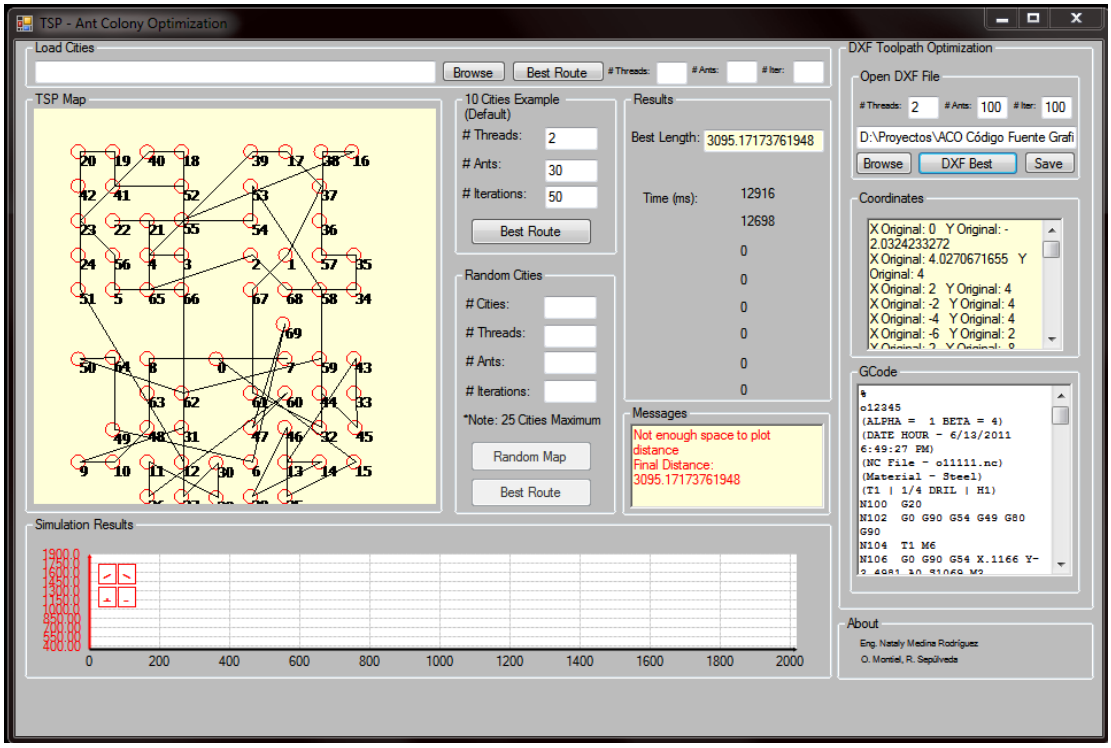

Figura 4.7 Resultado gráfico efectuado por el experimento 4.

La figura 4.7 muestra el cuarto experimento para las pruebas del algoritmo basado en cómputo paralelo, generando la trayectoria optimizada de la herramienta de taladrado como se muestra en el mapa de resultados visuales. Las tablas 4.10, 4.11 y 4.12 muestran los resultados experimentales correspondientes relacionados al tiempo de ejecución del algoritmo de acuerdo al número de procesadores utilizados; además se determina la ganancia de velocidad y la eficiencia del algoritmo.

Tabla 4.10 Resultados experimento 4 – A.

65 nodos, 100 hormigas y 100 iteraciones, alpha 1, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	25116	-	-
2	20478	1.226487	0.613243481
3	9857	2.548037	0.849345643
4	7058	3.558515	0.88962879
5	5604	4.481799	0.896359743
6	4893	5.133047	0.855507868

Tabla 4.11 Resultados experimento 4 – B.

65 nodos, 100 hormigas y 100 iteraciones, alpha 2, beta 4.5			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	18174	-	-
2	20478	0.887489	0.44374451
3	10789	1.6844935	0.56149782
4	7058	2.5749504	0.6437376
5	5604	3.2430407	0.64860814
6	3569	5.0921827	0.84869711

Tabla 4.12 Resultados experimento 4 – C.

65 nodos, 100 hormigas y 100 iteraciones, alpha 2.5, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	18033	-	-
2	12689	1.421152	0.71057609
3	9899.5	1.821607	0.607202384
4	7058	2.554973	0.63874327
5	6589	2.736834	0.547366823
6	3898	4.626219	0.771036429

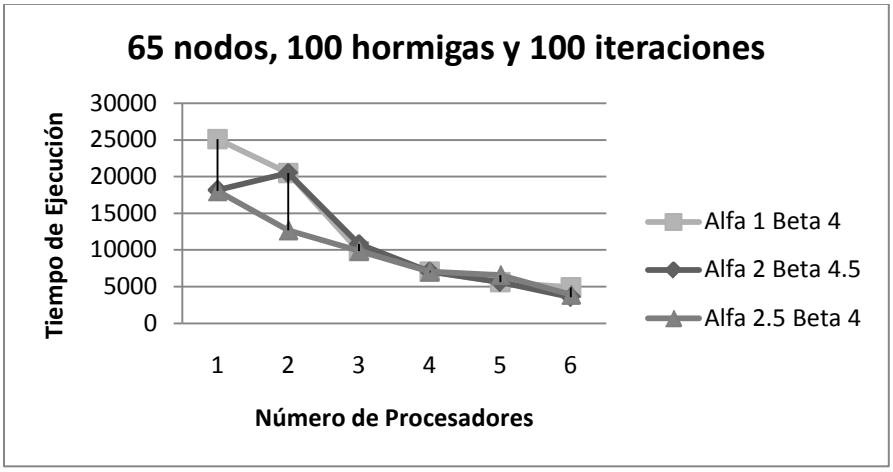

Figura 4.8 Tiempos de ejecución de acuerdo al número de procesadores para el experimento 4.

Para un problema de 65 nodos, la diferencia en la reducción de los tiempos de ejecución es notoria en el primer número de procesadores utilizados.

E. Experimento 5

La figura 4.9 muestra el quinto experimento para las pruebas del algoritmo basado en cómputo paralelo, generando la trayectoria optimizada de la herramienta de taladrado como se muestra en el mapa de resultados visuales. Las tablas 4.13, 4.14 y 4.15 muestran los resultados experimentales correspondientes relacionados al tiempo de ejecución del algoritmo de acuerdo al número de procesadores utilizados; además se determina la ganancia de velocidad y la eficiencia del algoritmo.

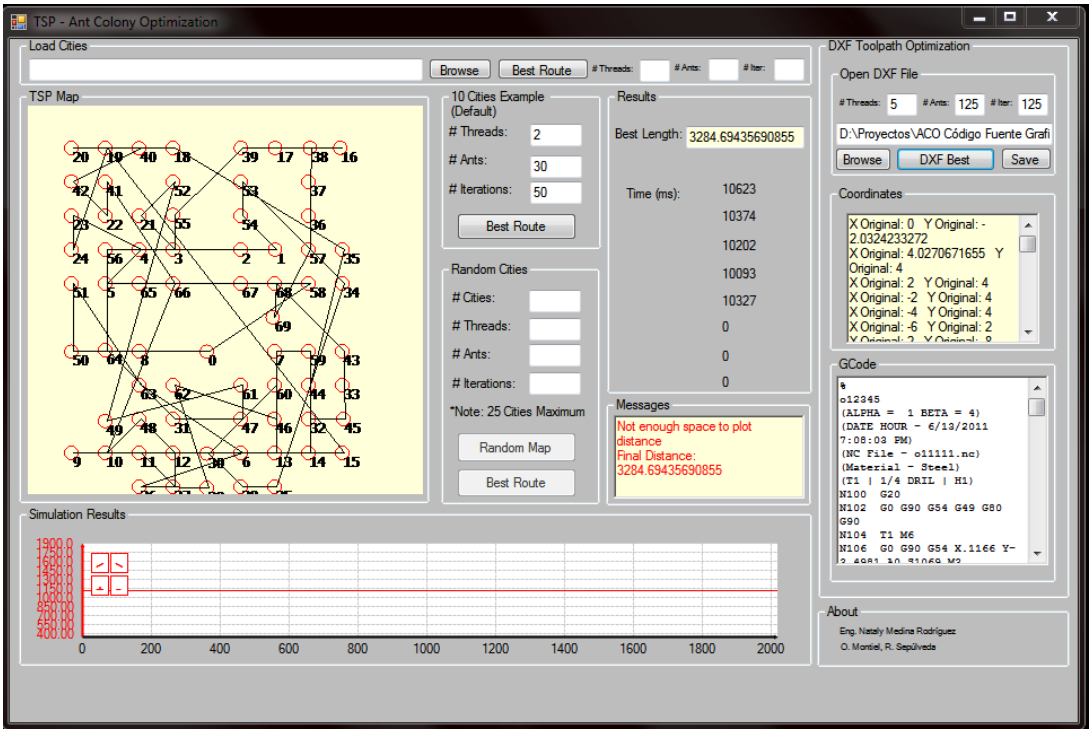

Figura 4.9 Resultado gráfico efectuado por el experimento 5.

Tabla 4.13 Resultados experimento 5 – A.

70 nodos, 125 hormigas y 125 iteraciones, alpha 1, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	39858	-	-
2	20478	1.946381	0.973190741
3	16789	2.374054	0.79135148
4	14878	2.678989	0.669747278
5	9872	4.03748	0.807495948
6	8789	4.534987	0.755831153

Tabla 4.14 Resultados experimento 5 – B.

70 nodos, 125 hormigas y 125 iteraciones, alpha 2, beta 4.5			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	39598	-	-
2	20456	1.9357646	0.96788228
3	18976	2.0867411	0.69558038
4	14798	2.6759021	0.66897554
5	8798	4.5007956	0.90015913
6	7897	5.0143092	0.83571821

Tabla 4.15 Resultados experimento 5 – C.

70 nodos, 125 hormigas y 125 iteraciones, alpha 2.5, beta 4			
Número de procesadores	Tiempo de Ejecución (ms)	Aceleración (*Speedup*)	Eficiencia
1	38578	-	-
2	20456	1.885901	0.942950724
3	18456	2.090269	0.696756249
4	14559	2.64977	0.662442475
5	8547	4.513631	0.902726103
6	6897	5.593446	0.932241071

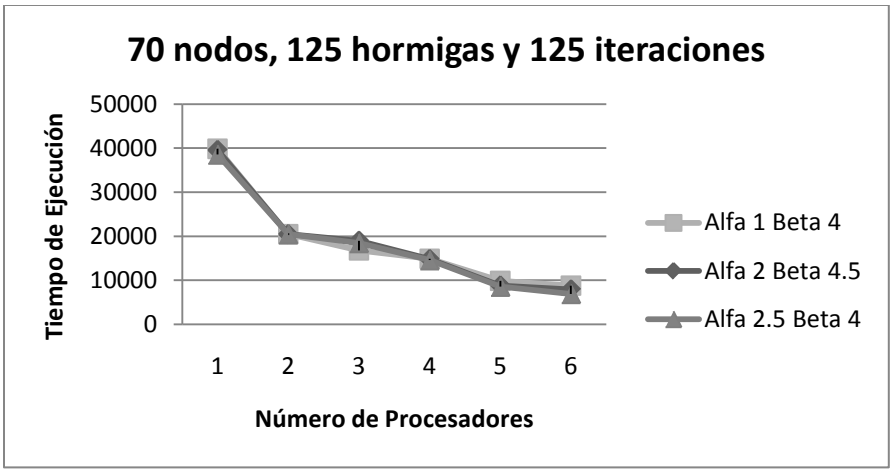

Figura 4.10 Tiempos de ejecución de acuerdo al número de procesadores para el experimento 5.

Para el quinto experimento que consistió en 70 nodos, la diferencia entre los parámetros de control entre las tres ejecuciones realizadas es mínima, sin embargo se puede observar que este experimento también muestra una reducción en los tiempos de ejecución del algoritmo proporcional al número de procesadores utilizados.

4.2 Análisis de resultados de tiempos de manufactura

La aplicación final del algoritmo ha sido implementada en una máquina de control numérico por computadora CNC HAAS Automation® instalada en los laboratorios de CETYS Universidad campus Tijuana (ver Figura 4.11); consta de tres ejes, unidad de control, una unidad de enfriamiento y control de herramientas.

Figura 4.11 Máquina CNC HAAS Automation®.

Para comparar los resultados generados entre el paquete comercial MasterCAM® y el algoritmo diseñado en el presente trabajo de investigación, se ha diseñado una serie de piezas de prueba para efectuar la manufactura de las mismas.

A. Maquinado 1

El maquinado 1 consiste en el proceso de taladrado de cinco piezas mostradas en las figuras 4.12 a 4.15, en placas de acero de ¼" con dimensiones de 8" x 8" pulgadas, mismas que se generan en consola la trayectoria óptima encontrada por el algoritmo así como la generación del código numérico para ser implementado en la máquina CNC.

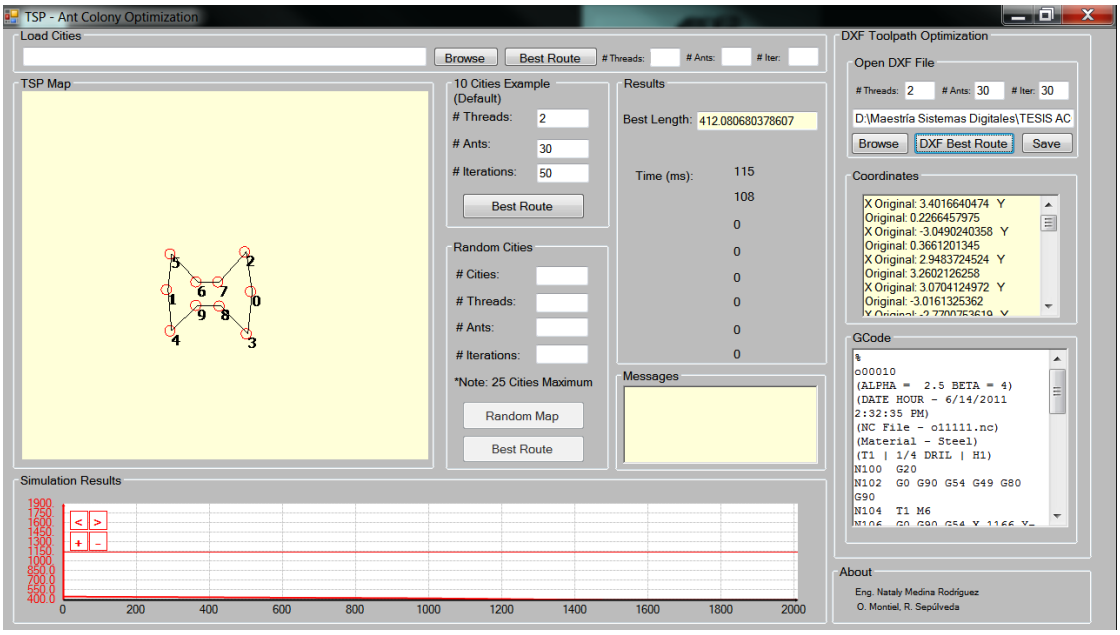

Figura 4.12 Generación de trayectoria para una pieza con 10 orificios.

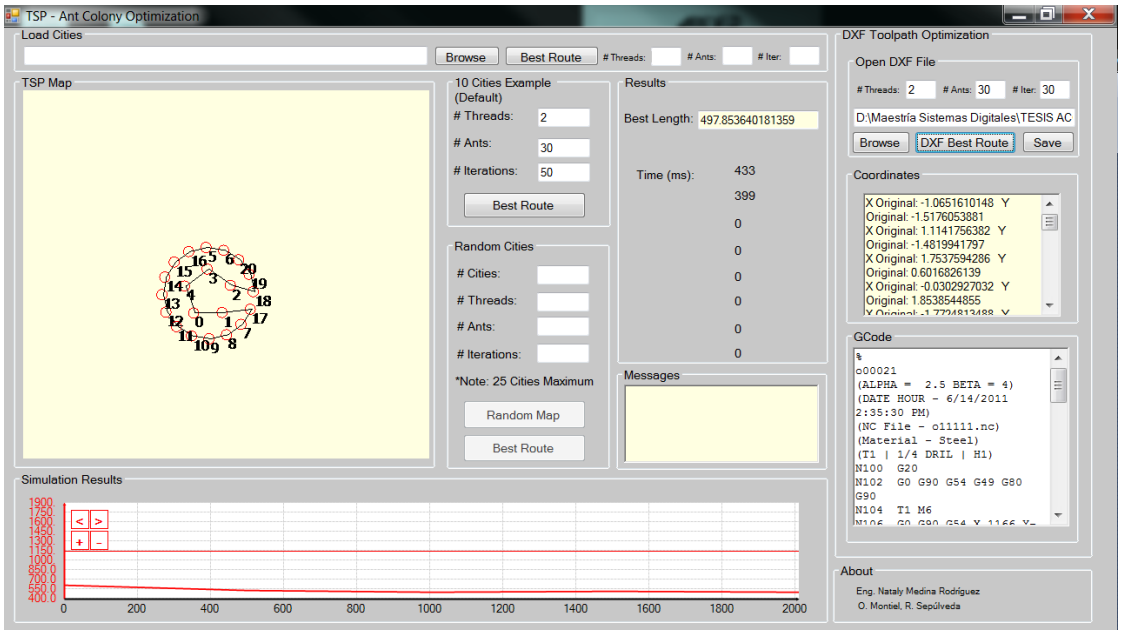

Figura 4.13 Generación de trayectoria para una pieza con 21 orificios.

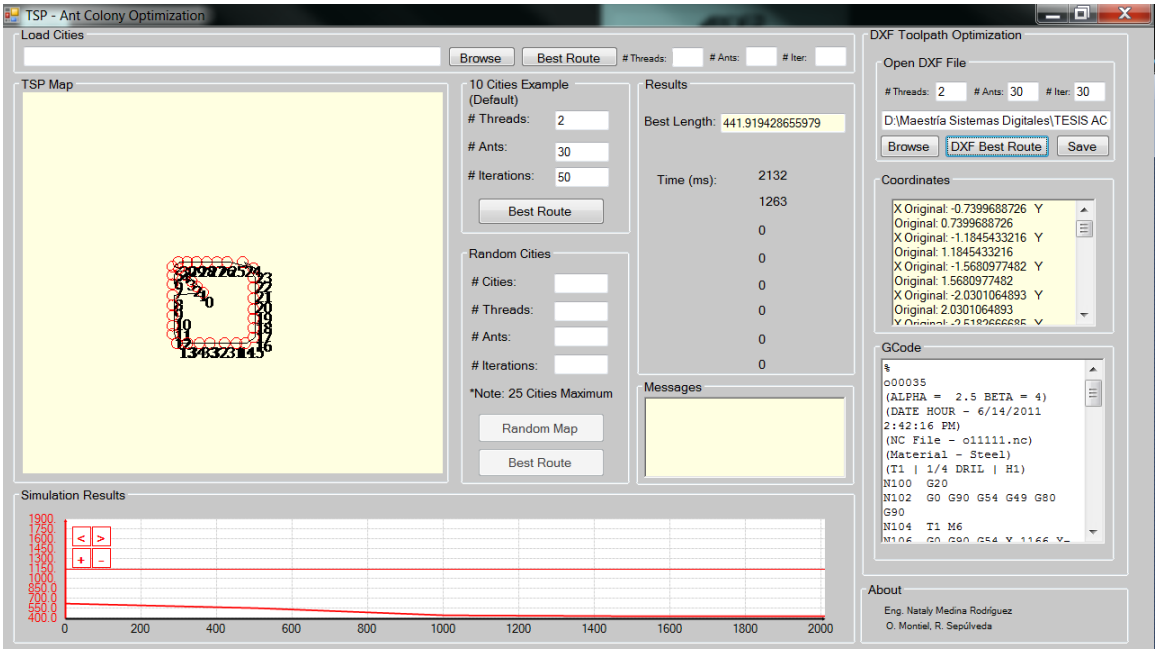

Figura 4.14 Generación de trayectoria para una pieza con 35 orificios.

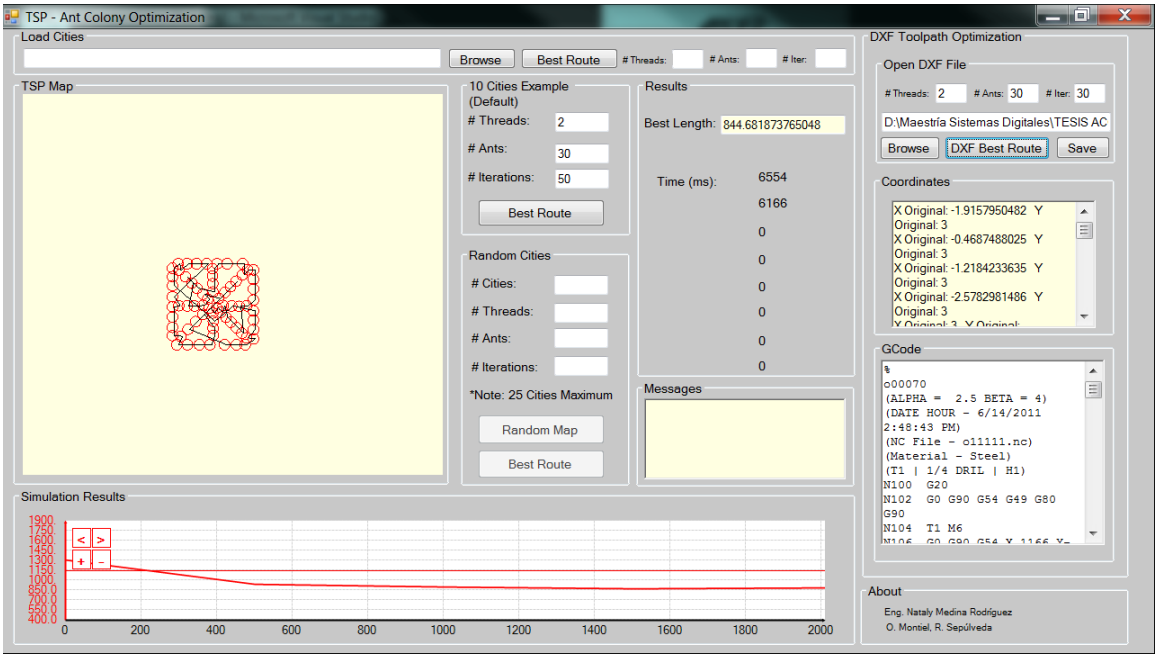

Figura 4.15 Generación de trayectoria para una pieza con 70 orificios.

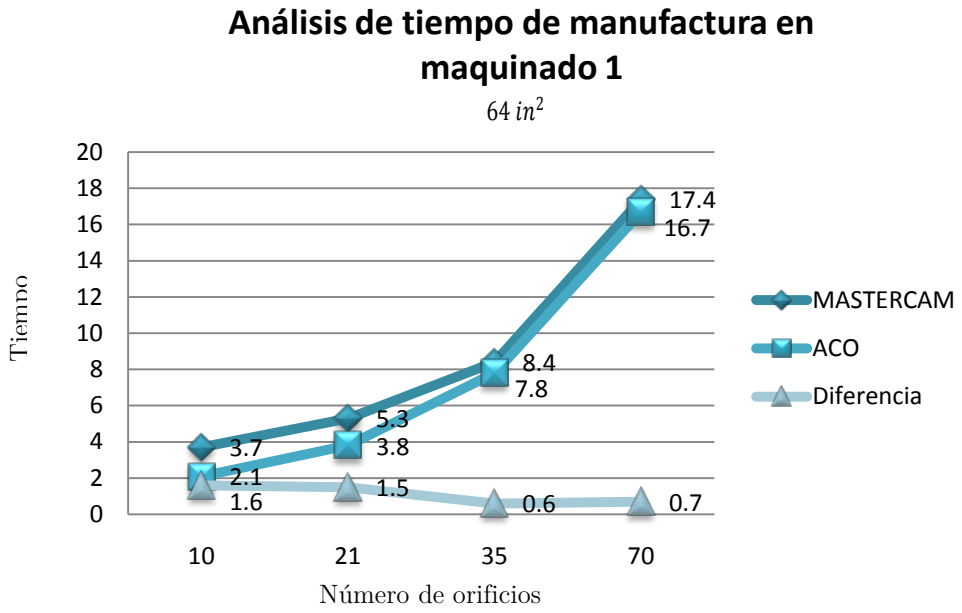

Figura 4.16 Tiempos de maquinado entre el diseño de un paquete comercial y el algoritmo implementado para una pieza con un área superficial de $64\ in^2$.

La figura 4.16 muestra la diferencia en los tiempos de maquinado, en este caso de taladrado de cinco piezas con diferente distribución de orificios y cantidad de los mismos.

B. Maquinado 2

El maquinado 2 consiste en el proceso de taladrado de cinco piezas mostradas en las figuras 4.17 a 4.20, en placas de acero de ¼" con dimensiones de 15" x 15" pulgadas, mismas que se generan en consola la trayectoria óptima encontrada por el algoritmo así como la generación del código numérico para ser implementado en la máquina CNC.

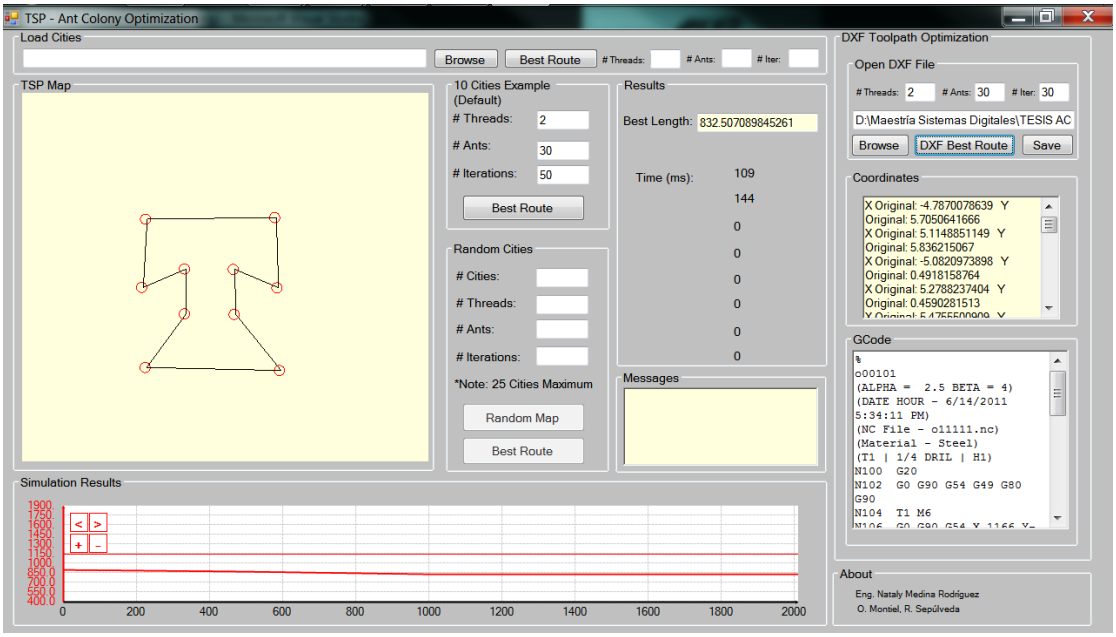

Figura 4.17 Generación de trayectoria para una pieza con 10 orificios.

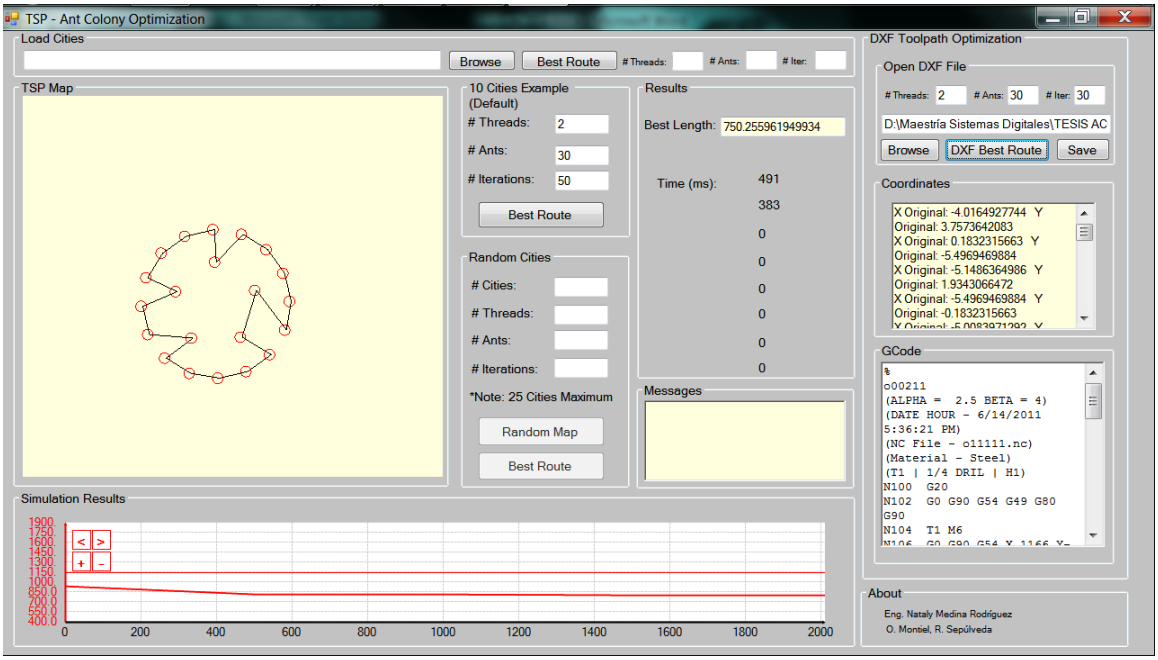

Figura 4.18 Generación de trayectoria para una pieza con 21 orificios.

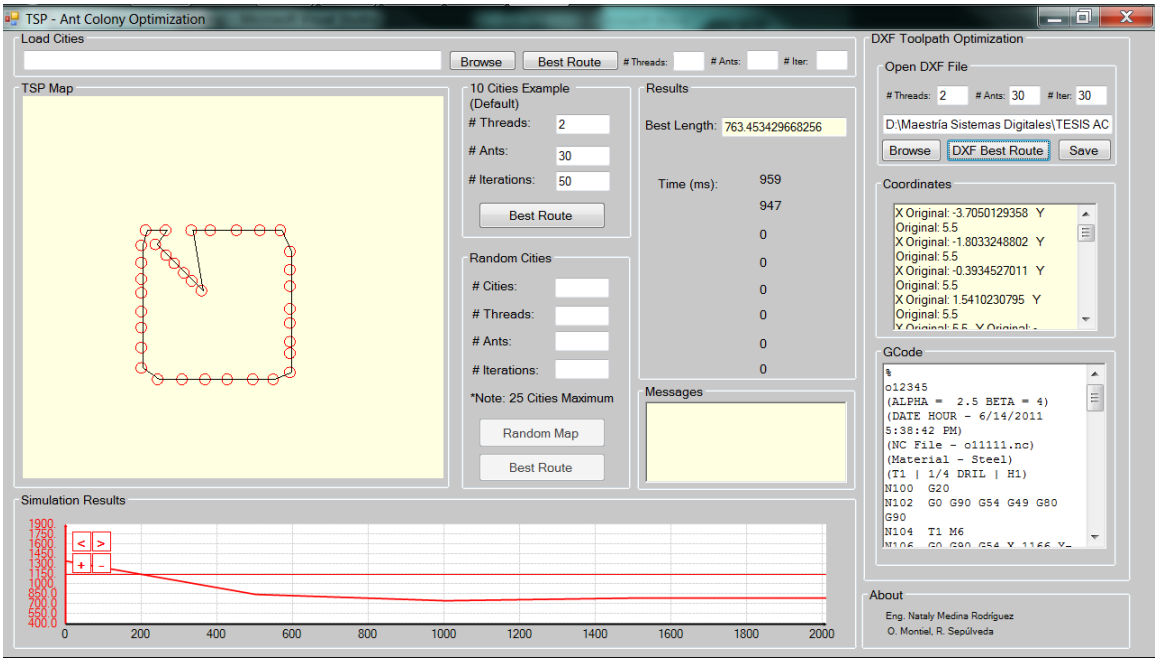

Figura 4.19 Generación de trayectoria para una pieza con 35 orificios.

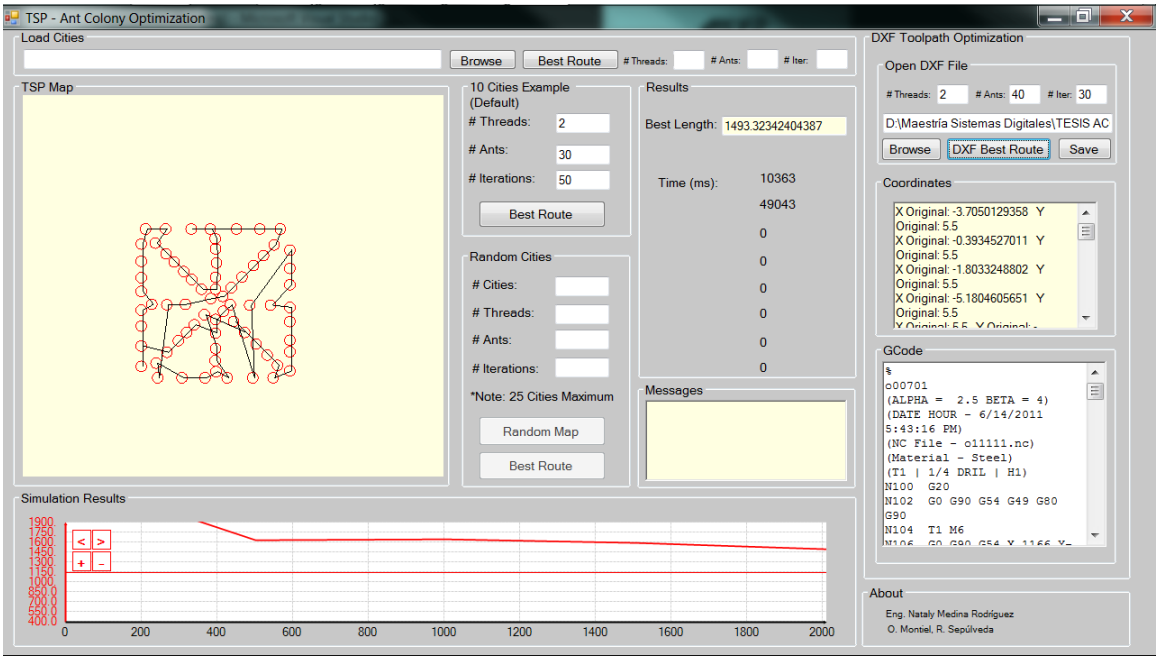

Figura 4.20 Generación de trayectoria para una pieza con 70 orificios.

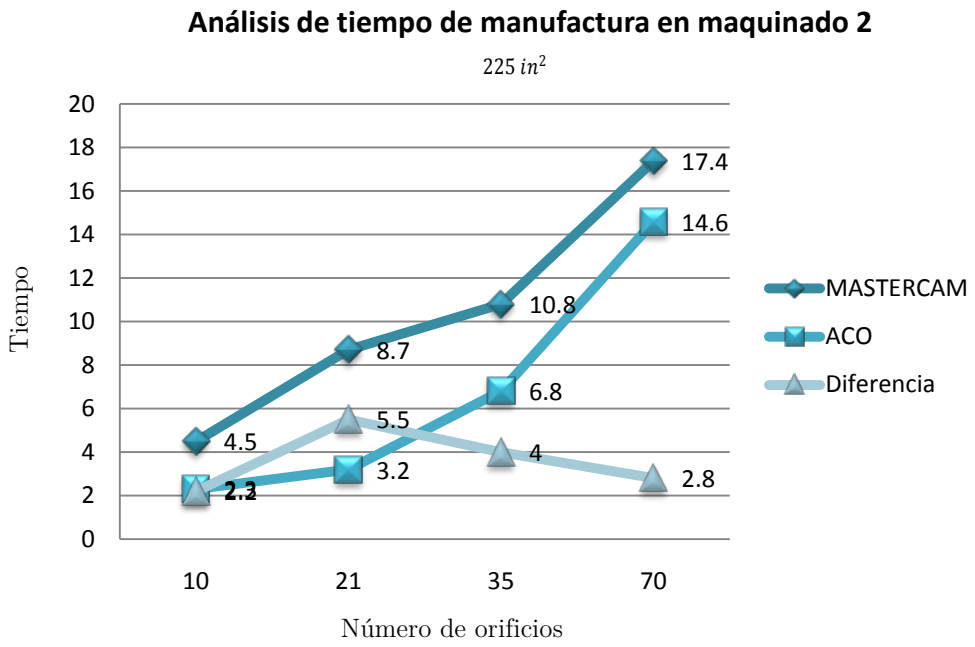

Figura 4.21 Tiempos de maquinado entre el diseño de un paquete comercial y el algoritmo implementado para una pieza con un área superficial de 225 in^2.

Se puede observar en estas dos pruebas realizadas que el aumento del área superficial incrementa la diferencia entre los algoritmos para la generación de trayectorias cuando la pieza presenta una cantidad lo suficientemente grande en comparación con el espacio vacío entre cada orificio.

Cabe destacar que MasterCAM® genera trayectorias aleatorias y por defecto son del tipo taladrado profundo o "*peck drill*"; también nos muestra la posibilidad de seleccionar una trayectoria en específico. Sin embargo, estas características han presentado una diferencia en el consumo de tiempo al comparar con las trayectorias generadas por el algoritmo de optimización mediante Colonia de Hormigas.

Para la aplicación implementada, al ser placas con un espesor mínimo de ¼", es posible no realizar taladrado profundo, esto consume tiempo de maquinado, ya que las piezas son lo suficientemente delgadas en relación al espesor.

Por lo que se puede concluir de acuerdo a los resultados obtenidos que para el taladrado de placas lo suficientemente delgadas, tales como placas PCB, el algoritmo da solución a la trayectoria que seguirá la herramienta de trabajo de una máquina de Control Numérico por Computadora, optimizando la ruta de acuerdo al principio Hamiltoniano, encontrando la ruta más corta en cuanto a distancia Euclideana, reduciendo tiempo de maquinado; además de que el algoritmo no utiliza un taladrado profundo.

Capítulo 5

Conclusiones

En el presente trabajo de tesis se desarrolló una plataforma de software basada en la tecnología Microsoft® .NET Framework implementada en Visual C# que tiene como aplicación final la optimización de trayectorias para máquinas de Control Numérico por Computadora utilizando metaheurísticas, en este caso, se ha utilizado la Optimización por Colonia de Hormigas para la solución del conocido Problema del Agente Viajero, un caso fundamental para llevar a cabo la aplicación desarrollada; además, el algoritmo se ha desarrollado mediante el uso de la computación paralela, haciendo uso de los recursos de hardware para la división de tareas, que para el caso del algoritmo, la ejecución de la tarea de la búsqueda de la solución k que cada hormiga encuentra, se calcula de acuerdo a los núcleos de la arquitectura del procesador.

El presente trabajo consistió en la descripción de las diferentes partes del desarrollo de la plataforma; en primera instancia, se implementó el algoritmo por Colonia de Hormigas conocido como Sistema de Hormigas, estructurando de manera secuencial la inicialización de todos aquéllos parámetros fundamentales del algoritmo, tales como la concentración de feromona, coeficiente de evaporación y parámetros de control de la visibilidad y heurística de la hormiga artificial;

posteriormente se comienza la ejecución en paralelo de las tareas más pesadas computacionalmente, esto es, la búsqueda de la solución por cada hormiga independiente, actualizando el rastro de feromona al encontrar la ruta óptima, sincronizando este último parámetro de manera independiente, es por ello que el problema presenta la capacidad de poder paralelizar las tareas. Al finalizar la sincronización, se actualiza el rastro de feromona y se evalúa la solución encontrada.

Una vez implementando el algoritmo, se procedió a interpretar archivos DXF que contienen la información de la pieza a maquinar, en este caso, la pieza se le realizará un taladrado. Cuando se interpreta el archivo, es ingresado al algoritmo para que proporcione una solución de trayectoria que llevará la herramienta de perforación. Finalmente, la plataforma de software genera un código numérico conocido como código G que se implementa directamente en una máquina de Control Numérico por Computadora o CNC.

Al efectuar pruebas para verificar la eficiencia del algoritmo, se destaca que el uso de las tecnologías multinúcleo permite reducir tiempos de ejecución del mismo, además de reducir la complejidad computacional. Esto hace que un algoritmo sea escalable, de manera tal que entre mayor número de procesadores y mayor número de trabajos o en este caso, mayor número de orificios, se presente una linealidad de la relación entre la ganancia de velocidad o *speedup* y el número de elementos de proceso n. El sistema será escalable mientras la relación no se encuentre por debajo de la linealidad.

En otras palabras, podemos decir que el algoritmo se evalúa con relación al incremento del tamaño tanto del sistema paralelo (cantidad de núcleos, memoria) como del problema. El algoritmo implementado en este trabajo de tesis presenta la característica de que el tiempo de ejecución se reduce proporcionalmente de acuerdo al número de procesadores utilizados, por lo que se puede afirmar que el algoritmo implementado es escalable.

Por otra parte, la optimización de las trayectorias mediante la optimización por Colonia de Hormigas han hecho posible la reducción del tiempo de maquinado para las piezas con espesores menores a ¼", tales como placas de PCB y similares. Esto puede formar parte de la filosofía de diseño y producción en la industria, optimizando así sus líneas de producción.

5.1 Trabajo a futuro

- Implementar el algoritmo de optimización utilizando las tecnologías emergentes, con el uso de GPUs para el cálculo de las tareas pesadas computacionalmente y CPU para el cómputo de la parte secuencial del algoritmo. Esto deberá generar mejores resultados en cuanto a la eficiencia del algoritmo.
- Implementar la optimización de trayectorias para taladrado en piezas no simétricas. Para lograr esto, se deberá optimizar la profundidad del agujero y la ubicación del mismo en un espacio tridimensional.
- Optimización de parámetros como el avance por minuto (*feedrate*) y el giro (*spindle*). Este tipo de solución deberá incluir la optimización multiobjetivo.

- Integrar a la plataforma de software una serie de interfaces gráficas para el usuario que le permita seleccionar la herramienta, las profundidades y el tipo de taladrado.
- Desarrollo de una interfaz para máquinas de taladrado del tipo no industrial, para efectuar trabajos educativos. Estas máquinas son conocidas en el ámbito educativo y generalmente no cuentan con algún software para la manipulación de las mismas.

Referencias

[1] P W Prickett, J Wang, *Expert system approach to CNC toolpath generation.* Factory 2000, Advanced Factory Automation Conf., 3-5 October, pp. pp 5-8., 1994

[2] Jaber E. Abu Qudeiri, Al-Momani Raid, Mohamed Anouar Jamali and Hidehiko Yamamoto, *Optimization Hole-Cutting Operations Sequence in CNC Machine Tools Using GA*. International Conference on Service systems and Service Management (ICSSSM06), PP.:501, 506, Troyes, France, Oct. 2006

[3] Abbas, A. T., Aly, M. F.,Hamza K, *Optimum drilling path planning for a rectangular matrix of holes using ant colony optimisation.* International Journal of Production Research, doi:10.1080/00207543.2010.507608, 2010.

[4] M. H. Alsuwaiyel, *Algorithms: Design Techniques and Analysis*, World Scientific Publishing Co. Pte. Ltd, 5 Toh Tuck Link, Singapore, 2004.

[5] Xin-She Yang, *Engineering Optimization. An Introduction with Metaheuristic Applications*, Wiley, 2010.

[6] Von Frisch K. *The Dance Language and Orientation of Bees*. Harvard University Press, Cambridge, 1967.

[7] Dušan Teodorović, Mauro Dell' Orco. *Bee Colony Optimization – A cooperative learning approach to complex transportation problems.* Advanced OR and AI Methods in Transportation,

[8] L. M. Gambardella M. Dorigo. Ant colonies for the traveling salesman problem. BioSystems, 43:73–81, 1997.

[9] Luca Maria Gambardella, Marco Dorigo: Solving Symmetric and Asymmetric TSPs by Ant Colonies. International Conference on Evolutionary Computation 1996: 622-627

[10] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[11] Marco Dorigo, Luca Maria Gambardella: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evolutionary Computation 1(1): 53-66, 1997.

[12] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29–41, 1996.

[13] Marco Dorigo, Gianni Di Caro, Luca Maria Gambardella: Ant Algorithms for Discrete Optimization. Artificial Life 5(2): 137-172, 1999.

[14] Pierre Delisle, Marc Gravel, Michaël Krajecki, Caroline Gagné, Wilson L. Price: Comparing Parallelization of an ACO: Message Passing vs. Shared Memory. Hybrid Metaheuristics 2005: pp. 1-11

[15] Blaise Barney, *Introduction to Parallel Computing.* Disponible en Lawrence Livermore National Laboratory: https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[16] R. Robec, M. Vajteršic, P. Zinterhof (Eds.) *Parallel computing: numerics, applications and trends.* Springer – Verlag, London 2009.

[17] Thomas Chen, Ram Raqhavan, Jason Date, Eiji Iwata. *Cell Boradband Engine Architecture and its first implementation.* Disponible en IBM: http://www.ibm.com/developerworks/power/library/pa-cellperf/

[18] Frank Mueller. *Sony PS3 Cluster (IBM Cell BE).* Disponible en NCSU: http://moss.csc.ncsu.edu/~mueller/cluster/ps3/

[19] S. Bell et al., *TILE64 Processor: A 64-Core SoC with Mesh Interconnect,* Digest of Technical Papers, IEEE International, 2008.

[20] S. Vangal et al., *An 80-Tile 1.28 TFLOPS Network-on-Chip in 65mm CMOS,* Digest of Technical Papers, IEEE International, 2007.

[21] A. Jantsch, H. Tenhunen (Eds.), *Network on Chip,* Kluwer Academic Publishers, Dortdrecht, 2003.

[22] Xilinx (2003), *Revolutionary architecture for the next generation platform FPGAs.* Disponible en Xilinx: http://www.xilinx.com/company/press/kits/asmbl/asmbl_arch_pres.pdf

[23] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, M. Schimmler, *A configuration concept for a massive parallel FPGA architecture, in:* International Conference on Computer Design (CDES'06), 2006.

[24] E. H. D' Hollander, D. Stroobandt, A. Touhafi, *Parallel computing with FPGAs – Concepts and applications,* in: Parallel Computing and Applicacionts, C. Bishof et al. (Eds.), NIC Series 38, 739-740, 2007.

[25] Baese, U. M. *Digital Signal Processing with Field Programmable Gate Arrays (Signals and Communication Technology).* Berlin, Germany: Springer, 2004.

[26] T. Wollinger, Ch. Paar, *How secure are FPGAs in cryptographic applications,* in: Proceedings of International Conference on Field Programmable Logic and Applications (FPL 2003), LNCS 2778, Springer – Verlag, Berlin 91 -100, 2003.

[27] NVIDIA (2011). *What is GPU Computing?* Disponible en NVIDIA: http://www.nvidia.com/object/GPU_Computing.html

[28] H. S. Bawa. *Manufacturing Processes – I.* McGraw-Hill Education, 2006.

[29] Peter Smid. *A comprehensive Guide to Practical CNC Programming.* Industrial Press Inc. Second Edition, 2000.

[30] A. Leyensetter, G. Würtemberger, Carlos Sáenz de Magarola. *Tecnología de los orificios metalúrgicos.* Editorial Reverté, S.A. 2006.

[31] Francisco Cruz Teruel. *Control Numérico y Programación, Sistemas de Fabricación de máquinas automatizadas.* Editorial Marcombo, S.A., 2004.

[32] Ali K. Kamrani, Emad Abouel Nasr. *Collaborative Engineering: Theory and Practice.* Springer Science + Business Media, LLC, 2008.

[33] P. Radhakrishnan, S. Subramanyan, V. Raju. *CAD/CAM/CIM.* New Age International Publishers, 2000.

[34] Autodesk (n.d). *DXF Group Codes.* Disponible en Autodesk: http://www.autodesk.com/techpubs/autocad/acadr14/dxf/dxf_group_codes.htm

[35] Thompson, W. T. *FITS.* Disponible en the FITS Support Office: NASA/GASFC: http://fits.gsfc.nasa.gov/wcs/coordinates.pdf

[36] Microsoft (n.d). *Coordinate Systems (Direct 3D 10).* Disponible en MSDN: http://msdn.microsoft.com/en-us/library/cc308049(v=vs.85).aspx

[37] Sourceforge.net (n.d). *Internet programming with Java Course.* Retrieved from Sourcefoge: http://inetjava.sourceforge.net/lectures/part2_applets/InetJava-2.1-2.2-Introduction-to-AWT-and-Applets.html

[38] Ling Chen, Hai-Ying, & Shu Wang, Parallel implementation of ant colony optimization on MPP. *Proc. International Conference on Machine Learning and Cybernetics,* Boading, 2008.

[39] Gao Dongdong, Gong Guanghong, Han Liang, Li Ni, Application of Multi-Core Parallel Ant Colony Optimization in Target Assignment Problem. *International Conference on Computer Application and System Modeling,* ICCASM, 2010.

[40] Jeremy Kepner, Parallel Matlab for Multicore and Multinode Computers. SIAM, 2009.

Apéndice I. **Código G**

G00 Rapid move G0 X# Y# Z# up to eight axes or G0 Z# X#

G01 Feed Rate move G1 X# Y# Z# up to eight axes or G1 Z# X#

G02 Clockwise move

G03 Counter Clockwise move

G04 Dwell time G04 L#

G08 Spline Smoothing On

G09 Exact stop check, Spline Smoothing Off

G10 A linear feedrate controlled move with a decelerated stop

G11 Controlled Decel stop

G17 XY PLANE

G18 XZ PLANE

G19 YZ PLANE

G28 Return to clearance plane

G33 Threading (Lathe)

G35 Bypass error checking on next line

G40 Tool compensation off

G41 Tool compensation to the left

G42 Tool compensation to the right

G43 Tool length compensation - negative direction

G44 Tool length compensation - positive direction

G49 Tool length compensation cancelled

G53 Cancel work coordinate offsets

G54-G59 Work coordinate offsets 1 through 6

G61 Spline contouring with buffering mode off

G64 Spline contouring with buffering mode on

G65 Mill out rectangular pocket

G66 Mill out circular pocket

G67 Flycut

G68 Mill out rectangular pocket with radius corners

G70 Inch mode

G71 Millimeter mode

G74 Peck drilling (Lathe) G83 Z# X# R#

G81 Drill cycle G81 X# Y# Z# R#

G82 Dwell cycle G82 X# Y# Z# R#

G83 Peck cycle G83 X# Y# Z# R#

G84 Tapping cycle G84 X# Y# Z# R# C#

G85 Boring cycle 1 G85 X# Y# Z# R#

G86 Boring cycle 2 G86 X# Y# Z# R#

G88 Boring cycle 3 G88 X# Y# Z# R#

G89 Boring cycle 4 G89 X# Y# Z# R#

G90 Absolute mode

G91 Incremental mode

G92 Home coordinate reset G92 X# Y# Z#

G94 IPM mode (Lathe) default

G95 IPR mode (Lathe)

G96 Constant Surface Feed On (Lathe)

G97 Constant Surface Feed Off (Lathe)

G110 Lathe Groove Face

G111 Lathe Groove OD

G112 Lathe Groove ID

G113 Lathe Thread OD

G114 Lathe Thread ID

G115 Lathe Face Rough

G116 Lathe Turn Rough

G120 Mill Outside Square

G121 Mill Outside Circle or Island

G122 Mill Out Counter Bore

G123 Mill Outside Ellipse pocket

G124 Mill Inside Ellipse pocket

G125 Mill Outside Slot

G126 Mill Inside Slot pocket

G130 3D tool compensation with gouge protection

G131 3D offset parallel to 3D profile

G132 3D tool compensation with gouge protection in the Z axis only

G135 5 axis tool compensation with gouge protection

G136 Included angle limit for gouge protection. G136 L#

G140 3D part rotation and plane tilting G140 U# V# W# R#

G141 Scale factor for the X axis only. G141 L#

G142 Scale factor for the Y axis only. G142 L#

G143 Scale factor for the Z axis only. G143 L#

G160 Mill 3D Cylinder

G162 Mill 3D Sphere

G163 Mill 3D Ramped Plane

G170 Set soft limits and crash fixture/chuck barriers to

defaults

G171 Set backward crash fixture/chuck barriers G171 U# V# W#

G172 Set forward crash fixture/chuck barriers G172 U# V# W#

G181 Bolt Hole Drill

G182 Bolt Hole Dwell

G183 Bolt Hole Peck

G184 Bolt Hole TapG185 Bolt Hole Bore

Apéndice II. **Código M**

M02 End of Program

M03 Spindle On Clockwise, Laser, Flame, Power ON

M04 Spindle On Counter Clockwise

M05 Spindle Stop, Laser, Flame, Power OFF

M06 Tool Change

M08 Coolant On

M09 Coolant Off

M10 Reserved for tool height offset

M13 Spindle On, Coolant On

M30 End of Program when macros are used

M91 Readout Display Incremental

M92 Readout Display Absolute

M97 Go to or jump to line number

M98 Jump to macro or subroutine

M99 Return from macro or subroutine

M100 Machine Zero Reset

M199 Mid program start

Apéndice III. **Grupos de Códigos DXF**

Group code ranges

Code range	**Group value type**
0-9	String (255 characters maximum; less for Unicode strings)
10-59	Double precision 3D point
60-79	16-bit integer value
90-99	32-bit integer value
100	String (255 characters maximum; less for Unicode strings)
102	String (255 characters maximum; less for Unicode strings)
105	String representing hexadecimal (hex) handle value
140-147	Double precision scalar floating-point value
170-175	16-bit integer value
280-289	8-bit integer value
300-309	Arbitrary text string
310-319	String representing hex value of binary chunk
320-329	String representing hex handle value
330-369	String representing hex object IDs
999	Comment (string)
1000-1009	String (255 characters maximum; less for Unicode strings)
1010-1059	Floating-point value
1060-1070	16-bit integer value
1071	32-bit integer value

Entity group codes by number

Group code	**Description**
-5	APP: persistent reactor chain
-4	APP: conditional operator (used *only* with **ssget**)
-3	APP: extended data (XDATA) sentinel (fixed)
-2	APP: entity name reference (fixed)
-1	APP: entity name. This changes each time a drawing is opened. It is never saved. (fixed)
0	Text string indicating the entity type (fixed)
1	Primary text value for an entity
2	Name (attribute tag, block name, and so on)
3-4	Other textual or name values
5	Entity handle. Text string of up to 16 hexadecimal digits (fixed)
6	Linetype name (fixed)
7	Text style name (fixed)
8	Layer name (fixed)
9	DXF: variable name identifier (used only in HEADER section of the DXF file).
10	Primary point. This is the start point of a line or text entity, center of a circle, and so on. DXF: *X* value of the primary point (followed by *Y* and *Z* value codes 20 and 30) APP: 3D point (list of three reals)
11-18	Other points. DXF: *X* value of other points (followed by *Y* value codes 21-28 and *Z* value codes 31-38) APP: 3D point (list of three reals)
20, 30	DXF: *Y* and *Z* values of the primary point
21-28,	DXF: *Y* and *Z* values of other points

31-37	
38	DXF: entity's elevation if nonzero.
39	Entity's thickness if nonzero (fixed)
40-48	Floating-point values (text height, scale factors, and so on)
48	Linetype scale. Floating-point scalar value. Default value is defined for all entity types.
49	Repeated floating-point value. Multiple 49 groups may appear in one entity for variable-length tables (such as the dash lengths in the LTYPE table). A 7*x* group always appears *before* the first 49 group to specify the table length.
50-58	Angles (output in degrees to DXF files and radians through AutoLISP and ARX applications).
60	Entity visibility. Integer value. Absence or 0 indicates visibility; 1 indicates invisibility.
62	Color number (fixed)
66	"Entities follow" flag (fixed)
67	Space--that is, model or paper space (fixed)
68	APP: identifies whether viewport is on but fully off screen; is not active or is off.
69	APP: viewport identification number.
70-78	Integer values, such as repeat counts, flag bits, or modes
90-99	32-bit integer values
100	Subclass data marker (with derived class name as a string). Required for all objects and entity classes that are derived from another concrete class to segregate data defined by different classes in the inheritance chain for the same object. This is in addition to the requirement for DXF names for each distinct concrete class derived from ARX (see "Subclass Markers").
102	Control string, followed by "{<arbitrary name>" or "}". Similar to the xdata 1002 group code, except that when the string begins with "{", it can be followed by an arbitrary string whose interpretation is up to the application. The only other allowable control string is

	"}" as a group terminator. As noted before, AutoCAD does not interpret these strings except during drawing audit operations; they are for application use.
105	DIMVAR symbol table entry object handle
210	Extrusion direction (fixed). DXF: *X* value of extrusion direction APP: 3D extrusion direction vector
220, 230	DXF: *Y* and *Z* values of the extrusion direction
280-289	8-bit integer values
300-309	Arbitrary text strings
310-319	Arbitrary binary chunks with same representation and limits as 1004 group codes: hexadecimal strings of up to 254 characters represent data chunks of up to 127 bytes.
320-329	Arbitrary object handles. Handle values that are taken "as is." They are not translated during INSERT and XREF operations.
330-339	Soft-pointer handle. Arbitrary soft pointers to other objects within same DXF file or drawing. Translated during INSERT and XREF operations.
340-349	Hard-pointer handle. Arbitrary hard pointers to other objects within same DXF file or drawing. Translated during INSERT and XREF operations.
350-359	Soft-owner handle. Arbitrary soft ownership links to other objects within same DXF file or drawing. Translated during INSERT and XREF operations.
360-369	Hard-owner handle. Arbitrary hard ownership links to other objects within same DXF file or drawing. Translated during INSERT and XREF operations.
999	DXF: The 999 group code indicates that the line following it is a comment string. DXFOUT does not include such groups in a DXF output file, but DXFIN honors them and ignores the comments. You can use the 999 group to include comments in a DXF file that you've edited.
1000	ASCII string (up to 255 bytes long) in extended data.
1001	Registered application name (ASCII string up to 31 bytes long) for extended data.

1002	Extended data control string ("{"or "}").
1003	Extended data layer name.
1004	Chunk of bytes (up to 127 bytes long) in extended data.
1005	Entity handle in extended data. Text string of up to 16 hexadecimal digits
1010	A point in extended data DXF: *X* value (followed by 1020 and 1030 groups) APP: 3D point
1020, 1030	DXF*: Y* and *Z* values of a point
1011	A 3D world space position in extended data DXF*: X* value (followed by 1021 and 1031 groups) APP: 3D point
1021, 1031	DXF: *Y* and *Z* values of a World space position
1012	A 3D world space displacement in extended data DXF: *X* value (followed by 1022 and 1032 groups) APP: 3D vector
1022, 1032	DXF: *Y* and *Z* values of a World space displacement
1013	A 3D world space direction in extended data. DXF: *X* value (followed by 1022 and 1032 groups) APP: 3D vector
1023, 1033	DXF: *Y* and *Z* values of a World space direction
1040	Extended data floating-point value.
1041	Extended data distance value.
1042	Extended data scale factor.

1070	Extended data 16-bit signed integer.
1071	Extended data 32-bit signed long.

The following is an example of the HEADER section of a DXF file:

0 SECTION 2 HEADER	*Beginning of HEADER section*
9 $<variable> <group code> <value>	*Repeats for each header variable*
0 ENDSEC	*End of HEADER section*

DXF system variables

Variable	**Group code**	**Description**
$ACADMAINTVER	70	Maintenance version number (should be ignored)
$ACADVER	1	The AutoCAD drawing database version number: AC1006 = R10, AC1009 = R11 and R12, AC1012 = R13, AC1014

		= R14
$ANGBASE	50	Angle 0 direction
$ANGDIR	70	1 = clockwise angles, 0 = counterclockwise
$ATTDIA	70	Attribute entry dialogs: 1 = on, 0 = off
$ATTMODE	70	Attribute visibility: 0 = none, 1 = normal, 2 = all
$ATTREQ	70	Attribute prompting during INSERT: 1 = on, 0 = off
$AUNITS	70	Units format for angles
$AUPREC	70	Units precision for angles
$BLIPMODE	70	Blip mode on if nonzero
$CECOLOR	62	Current entity color number: 0 = BYBLOCK, 256 = BYLAYER
$CELTSCALE	40	Current entity linetype scale
$CELTYPE	6	Entity linetype name, or BYBLOCK or BYLAYER
$CHAMFERA	40	First chamfer distance
$CHAMFERB	40	Second chamfer distance
$CHAMFERC	40	Chamfer length
$CHAMFERD	40	Chamfer angle
$CLAYER	8	Current layer name
$CMLJUST	70	Current multiline justification: 0=Top,1=Middle, 2=Bottom
$CMLSCALE	40	Current multiline scale
$CMLSTYLE	2	Current multiline style name
$COORDS	70	Coordinate display: 0 = static, 1 = continuous update, 2 = "*d*<*a*" format
$DELOBJ	70	Controls object deletion: 0=deleted, 1=retained
$DIMALT	70	Alternate unit dimensioning performed if nonzero
$DIMALTD	70	Alternate unit decimal places

$DIMALTF	40	Alternate unit scale factor
$DIMALTTD	70	Number of decimal places for tolerance values of an alternate units dimension
$DIMALTTZ	70	Controls suppression of zeros for alternate tolerance values: 0 = Suppresses zero feet and precisely zero inches 1 = Includes zero feet and precisely zero inches 2 = Includes zero feet and suppresses zero inches 3 = Includes zero inches and suppresses zero feet
$DIMALTU	70	Units format for alternate units of all dimension style family members except angular: 1 = Scientific; 2 = Decimal; 3 = Engineering; 4 = Architectural (stacked); 5 = Fractional (stacked); 6 = Architectural; 7 = Fractional
$DIMALTZ	70	Controls suppression of zeros for alternate unit dimension values: 0 = Suppresses zero feet and precisely zero inches 1 = Includes zero feet and precisely zero inches 2 = Includes zero feet and suppresses zero inches 3 = Includes zero inches and suppresses zero feet
$DIMAPOST	1	Alternate dimensioning suffix
$DIMASO	70	1 = create associative dimensioning, 0 = draw individual entities
$DIMASZ	40	Dimensioning arrow size
$DIMAUNIT	70	Angle format for angular dimensions: 0=Decimal degrees, 1=Degrees/minutes/seconds, 2=Gradians, 3=Radians, 4=Surveyor's units
$DIMBLK	1	Arrow block name
$DIMBLK1	1	First arrow block name
$DIMBLK2	1	Second arrow block name

$DIMCEN	40	Size of center mark/lines
$DIMCLRD	70	Dimension line color: range is 0 = BYBLOCK, 256 = BYLAYER
$DIMCLRE	70	Dimension extension line color: range is 0 = BYBLOCK, 256 = BYLAYER
$DIMCLRT	70	Dimension text color: range is 0 = BYBLOCK, 256 = BYLAYER
$DIMDEC	70	Number of decimal places for the tolerance values of a primary units dimension
$DIMDLE	40	Dimension line extension
$DIMDLI	40	Dimension line increment
$DIMEXE	40	Extension line extension
$DIMEXO	40	Extension line offset
$DIMFIT	70	Placement of text and arrowheads; Possible values: 0 through 3 (see appendix A, "System Variables," in the *Command Reference*)
$DIMGAP	40	Dimension line gap
$DIMJUST	70	Horizontal dimension text position: 0=above dimension line and center-justified between extension lines, 1=above dimension line and next to first extension line, 2=above dimension line and next to second extension line, 3=above and center-justified to first extension line, 4=above and center-justified to second extension line
$DIMLFAC	40	Linear measurements scale factor
$DIMLIM	70	Dimension limits generated if nonzero
$DIMPOST	1	General dimensioning suffix
$DIMRND	40	Rounding value for dimension distances
$DIMSAH	70	Use separate arrow blocks if nonzero
$DIMSCALE	40	Overall dimensioning scale factor
$DIMSD1	70	Suppression of first extension line:

		0=not suppressed, 1=suppressed
$DIMSD2	70	Suppression of second extension line: 0=not suppressed, 1=suppressed
$DIMSE1	70	First extension line suppressed if nonzero
$DIMSE2	70	Second extension line suppressed if nonzero
$DIMSHO	70	1 = Recompute dimensions while dragging, 0 = drag original image
$DIMSOXD	70	Suppress outside-extensions dimension lines if nonzero
$DIMSTYLE	2	Dimension style name
$DIMTAD	70	Text above dimension line if nonzero
$DIMTDEC	70	Number of decimal places to display the tolerance values
$DIMTFAC	40	Dimension tolerance display scale factor
$DIMTIH	70	Text inside horizontal if nonzero
$DIMTIX	70	Force text inside extensions if nonzero
$DIMTM	40	Minus tolerance
$DIMTOFL	70	If text outside extensions, force line extensions between extensions if nonzero
$DIMTOH	70	Text outside horizontal if nonzero
$DIMTOL	70	Dimension tolerances generated if nonzero
$DIMTOLJ	70	Vertical justification for tolerance values: 0=Top, 1=Middle, 2=Bottom
$DIMTP	40	Plus tolerance
$DIMTSZ	40	Dimensioning tick size: 0 = no ticks

$DIMTVP	40	Text vertical position
$DIMTXSTY	7	Dimension text style
$DIMTXT	40	Dimensioning text height
$DIMTZIN	70	Controls suppression of zeros for tolerance values: 0 = Suppresses zero feet and precisely zero inches 1 = Includes zero feet and precisely zero inches 2 = Includes zero feet and suppresses zero inches 3 = Includes zero inches and suppresses zero feet
$DIMUNIT	70	Units format for all dimension style family members except angular: 1 = Scientific; 2 = Decimal; 3 = Engineering; 4 = Architectural (stacked); 5 = Fractional (stacked); 6 = Architectural; 7 = Fractional
$DIMUPT	70	Cursor functionality for user positioned text: 0=controls only the dimension line location, 1=controls the text position as well as the dimension line location
$DIMZIN	70	Controls suppression of zeros for primary unit values: 0 = Suppresses zero feet and precisely zero inches 1 = Includes zero feet and precisely zero inches 2 = Includes zero feet and suppresses zero inches 3 = Includes zero inches and suppresses zero feet
$DISPSILH	70	Controls the display of silhouette curves of body objects in wire-frame mode: 0=Off, 1=On
$DRAGMODE	70	0 = off, 1 = on, 2 = auto
$DWGCODEPAGE	3	Drawing code page; Set to the system code page when a new drawing is created, but not otherwise maintained by AutoCAD
$ELEVATION	40	Current elevation set by ELEV command

$EXTMAX	10, 20, 30	*X*, *Y*, and *Z* drawing extents upper-right corner (in WCS)
$EXTMIN	10, 20, 30	*X*, *Y*, and *Z* drawing extents lower-left corner (in WCS)
$FILLETRAD	40	Fillet radius
$FILLMODE	70	Fill mode on if nonzero
$HANDLING	70	Next available handle
$HANDSEED	5	Next available handle
$INSBASE	10, 20, 30	Insertion base set by BASE command (in WCS)
$LIMCHECK	70	Nonzero if limits checking is on
$LIMMAX	10, 20	*XY* drawing limits upper-right corner (in WCS)
$LIMMIN	10, 20	*XY* drawing limits lower-left corner (in WCS)
$LTSCALE	40	Global linetype scale
$LUNITS	70	Units format for coordinates and distances
$LUPREC	70	Units precision for coordinates and distances
$MAXACTVP	70	Sets maximum number of viewports to be regenerated
$MEASUREMENT	70	Sets drawing units. 0=English; 1=Metric
$MENU	1	Name of menu file
$MIRRTEXT	70	Mirror text if nonzero
$ORTHOMODE	70	Ortho mode on if nonzero

$OSMODE	70	Running object snap modes
$PDMODE	70	Point display mode
$PDSIZE	40	Point display size
$PELEVATION	40	Current paper space elevation
$PEXTMAX	10, 20, 30	Maximum *X*, *Y*, and *Z* extents for paper space
$PEXTMIN	10, 20, 30	Minimum *X*, *Y*, and *Z* extents for paper space
$PICKSTYLE	70	Controls group selection and associative hatch selection: 0=No group selection or associative hatch selection, 1=Group selection, 2 =Associative hatch selection, 3 =Group selection and associative hatch selection
$PINSBASE	10, 20, 30	Paper space insertion base point
$PLIMCHECK	70	Limits checking in paper space when nonzero
$PLIMMAX	10, 20	Maximum *X* and *Y* limits in paper space
$PLIMMIN	10, 20	Minimum *X* and *Y* limits in paper space
$PLINEGEN	70	Governs the generation of linetype patterns around the vertices of a 2D polyline: 1 = linetype is generated in a continuous pattern around vertices of the polyline, 0 = each segment of the polyline starts and ends with a dash
$PLINEWID	40	Default polyline width
$PROXYGRAPHICS	70	Controls the saving of proxy object images
$PSLTSCALE	70	Controls paper space linetype scaling: 1 = no special linetype scaling

		0 = viewport scaling governs linetype scaling
$PUCSNAME	2	Current paper space UCS name
$PUCSORG	10, 20, 30	Current paper space UCS origin
$PUCSXDIR	10, 20, 30	Current paper space UCS *X* axis
$PUCSYDIR	10, 20, 30	Current paper space UCS *Y* axis
$QTEXTMODE	70	Quick text mode on if nonzero
$REGENMODE	70	REGENAUTO mode on if nonzero
$SHADEDGE	70	0 = faces shaded, edges not highlighted 1 = faces shaded, edges highlighted in black 2 = faces not filled, edges in entity color 3 = faces in entity color, edges in black
$SHADEDIF	70	Percent ambient/diffuse light, range 1-100, default 70
$SKETCHINC	40	Sketch record increment
$SKPOLY	70	0 = sketch lines, 1 = sketch polylines
$SPLFRAME	70	Spline control polygon display: 1 = on, 0 = of
$SPLINESEGS	70	Number of line segments per spline patch
$SPLINETYPE	70	Spline curve type for PEDIT Spline
$SURFTAB1	70	Number of mesh tabulations in first direction
$SURFTAB2	70	Number of mesh tabulations in second direction
$SURFTYPE	70	Surface type for PEDIT Smooth
$SURFU	70	Surface density (for PEDIT Smooth) in M direction
$SURFV	70	Surface density (for PEDIT Smooth) in N direction
$TDCREATE	40	Date/time of drawing creation

$TDINDWG	40	Cumulative editing time for this drawing
$TDUPDATE	40	Date/time of last drawing update
$TDUSRTIMER	40	User elapsed timer
$TEXTSIZE	40	Default text height
$TEXTSTYLE	7	Current text style name
$THICKNESS	40	Current thickness set by ELEV command
$TILEMODE	70	1 for previous release compatibility mode, 0 otherwise
$TRACEWID	40	Default trace width
$TREEDEPTH	70	Specifies the maximum depth of the spatial index.
$UCSNAME	2	Name of current UCS
$UCSORG	10, 20, 30	Origin of current UCS (in WCS)
$UCSXDIR	10, 20, 30	Direction of current UCS's X axis (in WCS)
$UCSYDIR	10, 20, 30	Direction of current UCS's Y axis (in WCS)
$UNITMODE	70	Low bit set = display fractions, feet-and-inches, and surveyor's angles in input format
$USERI1 - 5	70	Five integer variables intended for use by third-party developers
$USERR1 - 5	40	Five real variables intended for use by third-party developers
$USRTIMER	70	0 = timer off, 1 = timer on
$VISRETAIN	70	0 = don't retain xref-dependent visibility settings, 1 = retain xref-dependent visibility settings
$WORLDVIEW	70	1 = set UCS to WCS during DVIEW/VPOINT, 0 = don't change UCS

The following header variables existed prior to AutoCAD Release 11 but now have independent settings for each active viewport. DXFIN honors these variables when read from DXF files, but if a VPORT symbol table with *ACTIVE entries is present (as is true for any DXF file produced by Release 11 or higher), the values in the VPORT table entries override the values of these header variables.

Revised VPORT header variables

Variable	Group code	Description
$FASTZOOM	70	Fast zoom enabled if nonzero
$GRIDMODE	70	Grid mode on if nonzero
$GRIDUNIT	10, 20	Grid *X* and *Y* spacing
$SNAPANG	50	Snap grid rotation angle
$SNAPBASE	10, 20	Snap/grid base point (in UCS)
$SNAPISOPAIR	70	Isometric plane: 0 = left, 1 = top, 2 = right
$SNAPMODE	70	Snap mode on if nonzero
$SNAPSTYLE	70	Snap style: 0 = standard, 1 = isometric
$SNAPUNIT	10, 20	Snap grid *X* and *Y* spacing
$VIEWCTR	10, 20	*XY* center of current view on screen
$VIEWDIR	10, 20, 30	Viewing direction (direction from target in WCS)
$VIEWSIZE	40	Height of view

The following is an example of the CLASSES section of a DXF file:

0 SECTION 2 CLASSES	*Beginning of CLASSES section*
0 CLASS 1 <class dxf record> 2 <class name> 3 <app name> 90 <flag> 280 <flag> 281 <flag>	*Repeats for each entry*
0 ENDSEC	*End of CLASSES section*

CLASSES section group codes

Group code	Description
0	Record type (CLASS). Identifies beginning of a CLASS record.
1	Class DXF record name. These should always be unique.
2	C++ class name. Used to bind with software that defines object class behavior. These are always unique.
3	Application name. Posted in Alert box when a class definition listed in this section is not currently loaded.
90	Proxy capabilities flag. Bit coded value that indicates the capabilities of this object as a proxy. 0 = No operations allowed (0) 1 = Erase allowed (0x1) 2 = Transform allowed (0x2) 4 = Color change allowed (0x4) 8 = Layer change allowed (0x8) 16 = Linetype change allowed (0x10) 32 = Linetype scale change allowed (0x20) 64 = Visibility change allowed (0x40) 127 = All operations except cloning allowed (0x7F) 128 = Cloning allowed (0x80) 255 = All operations allowed (0xFF) 32768 = R13 format proxy (0x8000)
280	Was-a-proxy flag. Set to 1 if class was not loaded when this DXF file was created, and 0 otherwise.
281	Is-an-entity flag. Set to 1 if class was derived from the AcDbEntity class and can reside

	in the BLOCKS or ENTITIES section. If 0, instances may appear only in the OBJECTS section.

Default class values

DXF Record Name **Code 1**	**C++ Class Name** **Code 2**	**Code** **90**	**Code** **280**	**Code** **281**
DICTIONARYVAR	AcDbDictionaryVar	0	0	0
HATCH	AcDbHatch	0	0	1
IDBUFFER	AcDbIdBuffer	0	0	0
IMAGE	AcDbRasterImage	127	0	1
IMAGEDEF	AcDbRasterImageDef	0	0	0
IMAGEDEF_REACTOR	AcDbRasterImageDefReactor	1	0	0
LAYER_INDEX	AcDbLayerIndex	0	0	0
LWPOLYLINE	AcDbPolyline	0	0	1
OBJECT_PTR	CAseDLPNTableRecord	1	0	0
OLE2FRAME	AcDbOle2Frame	0	0	1
RASTERVARIABLES	AcDbRasterVariables	0	0	0
SORTENTSTABLE	AcDbSortentsTable	0	0	0
SPATIAL_INDEX	AcDbSpatialIndex	0	0	0
SPATIAL_FILTER	AcDbSpatialFilter	0	0	0

The following is an example of the BLOCKS section of a DXF file:

0 SECTION 2 BLOCKS	*Beginning of BLOCKS section*
0 BLOCK 5 <handle> 100 AcDbEntity 8 <layer> 100 AcDbBlockBegin 2 <block name> 70 <flag> 10 <X value> 20 <Y value> 30 <Z value> 3	*Begins each block entry* *(a block entity definition)*

<block name> 1 <xref path>	
0 <entity type> . . <data> .	*One entry for each entity definition within the block*
0 ENDBLK 5 <handle> 100 AcDbBlockEnd	*End of each block entry* *(an endblk entity definition)*
0 ENDSEC	*End of BLOCKS section*

Block group codes

Group codes	**Description**
0	Entity type (BLOCK)
5	Handle

102	Start of application defined group "{*application_name*". For example, "{ACAD_REACTORS" indicates the start of the AutoCAD persistent reactors group
application-defined codes	Codes and values within the 102 groups are application-defined.
102	End of group, "}"
100	Subclass marker (AcDbEntity)
8	Layer name
100	Subclass marker (AcDbBlockBegin)
2	Block name
70	Block-type flags (bit coded values, may be combined): 1 = This is an anonymous block generated by hatching, associative dimensioning, other internal operations, or an application 2 = This block has attribute definitions 4 = This block is an external reference (xref) 8 = This block is an xref overlay 16 = This block is externally dependent 32 = This is a resolved external reference, or dependent of an external reference (ignored on input) 64 = This definition is a referenced external reference (ignored on input)
10	Base point. DXF: *X* value; APP: 3D point
20, 30	DXF: *Y* and *Z* values of base point
3	Block name
1	Xref path name (optional; present only if the block is an xref)

The following group codes apply to endblk objects.

Endblk group codes

Group codes	**Description**
0	Entity type (ENDBLK)
5	Handle
102	Start of application defined group "{*application_name*". For example, "{ACAD_REACTORS" indicates the start of the AutoCAD persistent reactors group
application-defined codes	Codes and values within the 102 groups are application-defined.
102	End of group, "}"
100	Subclass marker (AcDbBlockEnd)

Group codes that apply to all graphical objects (ENTITIES)

Group code	**Description**	**If omitted, defaults to...**
-1	APP: entity name (changes each time a drawing is opened)	*Not* omitted
0	Entity type	*Not* omitted

5	Handle	*Not* omitted
102	Start of application defined group "{*application_name*". For example, "{ACAD_REACTORS" indicates the start of the AutoCAD persistent reactors group	no default
application-defined codes	Codes and values within the 102 groups are application-defined.	no default
102	End of group, "}"	no default
100	Subclass marker (AcDbEntity)	*Not* omitted
67	Absent or zero indicates entity is in model space. One indicates entity is in paper space. (optional)	0
8	Layer name	*Not* omitted
6	Linetype name (present if not BYLAYER). The special name BYBLOCK indicates a floating linetype. (optional)	BYLAYER
62	Color number (present if not BYLAYER). Zero indicates the BYBLOCK (floating) color. 256 indicates BYLAYER. A negative value indicates that the layer is turned off. (optional)	BYLAYER
48	Linetype scale (optional)	1.0
60	Object visibility (optional): 0 = visible, 1 = invisible.	0

The following table shows the group codes that are output if persistent reactors have been attached to an entity.

ACAD_REACTORS records

Group code	Description
102	"{ACAD_REACTORS" indicates the start of the AutoCAD persistent reactors group
330	Soft pointer ID/handle to owner dictionary
102	End of group, "}"

The following table shows the group codes that are output if an extension dictionary has been attached to an entity.

ACAD_XDICTIONARY records

Group code	Description
102	"{ACAD_XDICTIONARY" indicates the start of an extension dictionary group
360	Hard owner ID/handle to owner dictionary
102	End of group, "}"

The following group codes apply to line entities.

Line group codes

Group codes	Description
100	Subclass marker (AcDbLine)
39	Thickness (optional; default = 0)
10	Start point (in WCS). DXF: *X* value; APP: 3D point

20, 30	DXF: *Y* and *Z* values of start point (in WCS)
11	End point (in WCS). DXF: *X* value; APP: 3D point
21, 31	DXF: *Y* and *Z* values of end point (in WCS)
210	Extrusion direction. (optional; default = 0, 0, 1). DXF: *X* value; APP: 3D vector
220, 230	DXF: *Y* and *Z* values of extrusion direction

The following group codes apply to circle entities.

Circle group codes

Group codes	**Description**
100	Subclass marker (AcDbCircle)
39	Thickness (optional; default = 0)
10	Center point (in OCS). DXF: *X* value; APP: 3D point
20, 30	DXF: *Y* and *Z* values of center point (in OCS)
40	Radius
210	Extrusion direction. (optional; default = 0, 0, 1). DXF: *X* value; APP: 3D vector
220, 230	DXF: *Y* and *Z* values of extrusion direction

The following group codes apply to arc entities.

Arc group codes

Group codes	**Description**
100	Subclass marker (AcDbCircle)
39	Thickness (optional; default = 0)

10	Center point (in OCS). DXF: *X* value; APP: 3D point
20, 30	DXF: *Y* and *Z* values of center point (in OCS)
40	Radius
100	Subclass marker (AcDbArc)
50	Start angle
51	End angle
210	Extrusion direction. (optional; default = 0, 0, 1). DXF: *X* value; APP: 3D vector
220, 230	DXF: *Y* and *Z* values of extrusion direction

[34]