Universidad Central “Marta Abreu” de Las Villas

Facultad de Matemática, Física y Computación

Trabajo en opción al título de
Master en Ciencias de la Computación

Aplicación de Métodos Heurísticos en la solución de problemas de configuración

Autora

Lic. Yailen Martínez Jiménez

Tutor

Dr. Rafael Estéban Bello Pérez

2007

Hago constar que este trabajo ha sido realizado en la facultad de Matemática, Física y Computación de la Universidad Central "Marta Abreu" de las Villas como parte de la culminación de los estudios de la Maestría en Ciencias de la Computación, autorizando a que el mismo sea actualizado por la institución para los fines que estime conveniente, tanto de forma total como parcial y que además no podrá ser presentado en eventos ni publicado sin previa autorización de la universidad.

_____

Firma del Autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro, y que el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe del Seminario
de Inteligencia Artificial

A mi familia y mis amigos,

en especial a mi abuela

Agradezco a todas las personas que han contribuido al desarrollo de este trabajo:

A mi familia por todo el apoyo que me ha brindado, especialmente a mi mamá por todos sus consejos, dedicación y paciencia.

Al Dr. Rafael Bello y la Dra. María Matilde García, por el apoyo y el tiempo dedicado.

A Gheisa y Juliett por su amistad incondicional.

A todas mis amistades por estar siempre conmigo en los momentos difíciles, el tiempo es corto y no quiero obviar nombres.

A todos mis colegas y amigos por sus sugerencias y colaboración.

Resumen

El objetivo general de la investigación consiste en el desarrollo de algoritmos para la solución de problemas de configuración utilizando un modelo de razonamiento funcional.

En el contenido del trabajo se expone el marco teórico-referencial de la investigación, enfatizando en los modelos existentes para resolver este tipo de problema, así como la ausencia de aplicación de métodos heurísticos en este campo, donde las características del problema, que es minimizar la cantidad de componentes que aparecen en la solución, hace necesario su uso cuando la complejidad del problema aumenta.

Es por esto que se estudia la aplicación de tres métodos heurísticos, Ascensión de Colinas, Algoritmos Genéticos y Sistema de Colonias de Hormigas, los cuales fueron modelados de acuerdo a las características del problema a resolver y su desempeño fue probado usando diferentes bases de datos de prueba. Se realizó un análisis de los parámetros a utilizar en cada caso, analizando distintas variantes de ejecución.

El estudio comparativo realizado con las distintas soluciones encontradas por cada uno de los algoritmos propuestos arrojó que los mejores resultados son obtenidos en todos los casos por el algoritmo Sistema de Colonias de Hormigas, perteneciente a la metaheurística Optimización basada en Colonias de Hormigas.

Abstract

The major goal of this research project is the development of algorithms for the solution of configuration problems by utilizing a functional reasoning model.

The contents of this document clearly exhibit the theoretical-referential framework of the research conducted, stressing on the existing models for solving this sort of problems as well as the lack of application of heuristic methods in the aforementioned field, wherein the problem's essentials (minimizing the number of components that make up the solution) makes indispensable their use as the problem's complexity rises.

For such reason, three heuristic methods (Hill Climbing, Genetic Algorithms and Ant Colony System) are brought up and carefully applied and fashioned according to the features of the problem to be solved. Their performance was tested by means of several test data bases. A survey on how to set up the parameters of each algorithm is included in the research work.

As an outcome of the comparative study carried out between the different solutions found by each of the suggested algorithms, it has been shown that the best results are attained in all cases by the Ant Colony System procedure, belonging to the Ant Colony Optimization metaheuristic.

Índice

Introducción

La configuración es una tarea cuyo objetivo es la síntesis, a partir de componentes existentes, de un nuevo objeto el cual tiene que poseer un conjunto requerido de características funcionales. Cada componente existente tiene su propio conjunto de características funcionales y cuando 2, 3 o más de ellos se configuran, las características funcionales de la estructura resultante pueden ser derivadas de las de sus componentes. La configuración ha sido tratada de distintas formas y con diferentes métodos, a través del razonamiento funcional, por ejemplo, se ha desarrollado el proceso de configuración como una búsqueda a través de un espacio de artefactos posibles, muy utilizado en problemas de ingeniería de diseño.

Razonamiento Funcional es un término colectivo usado para una variedad de técnicas y teorías que sirven para explicar y deducir las funciones de los artefactos. Las técnicas de razonamiento funcional usualmente ofrecen un esquema de representación para describir los artefactos y una interfaz para inferir y explicar las funciones de los mismos y cómo ellos pueden contribuir a la funcionalidad de un sistema [1] .

El razonamiento funcional ha sido estudiado en una gran variedad de disciplinas, por ejemplo, la filosofía, la biología, la sociología y la ingeniería del diseño; ha sido reforzado por técnicas de ciencias de la computación e inteligencia artificial y su resultado es aplicado en la ingeniería de diseño, la planificación y el aprendizaje, entre otros [2].

Los sistemas basados en razonamiento funcional varían principalmente dependiendo del área de estudio, los esquemas de representación de las estructuras o funciones, los datos iniciales (descripción formal de la estructura de los artefactos y/o comportamiento), el enfoque del estudio y los problemas particulares.

Originalmente, las teorías de razonamiento funcional fueron dedicadas a explicar la presencia de las entidades dentro de los sistemas que las contienen, llámese sistema a un organismo vivo, una organización o un artefacto diseñado; por ejemplo, en biología se trata de contestar preguntas tales como ¿Por qué las jirafas tienen el cuello largo?, y se trata de descubrir la función del corazón en el cuerpo humano. En sociología, el Razonamiento Funcional es usado para explorar las condiciones de existencia necesarias de un sistema social [2].

Uno de los campos en los que su aplicación ha cobrado auge es la ingeniería de diseño y una tendencia reciente en las investigaciones sobre razonamiento funcional en esta rama consiste en dividir y descomponer la función, propósito y conceptos de objetivo y especificarlos en detalle. Similar a otras áreas de investigación, los desarrolladores de sistemas basados en razonamiento funcional usualmente comienzan definiendo su propio conjunto de conceptos fundamentales y

construyen su tecnología y prototipos sobre él, esto ha conducido a múltiples interpretaciones, solapamiento y conflictos parciales en las técnicas de razonamiento funcional [3].

Un sistema basado en razonamiento funcional es una implementación de una o más teorías de RF en un programa de computadora. Un sistema basado en RF típico incorpora lo siguiente:

- Mecanismos de representación de funciones y/o objetivos o propósitos. Por ejemplo, una base de datos de artefactos y sus funciones.
- Mecanismos de descripción de estado, estructura o comportamiento. Por ejemplo, una herramienta de simulación basada en el modelo.
- Mecanismos de inferencia para obtener funciones basados en una teoría de Razonamiento Funcional.
- Mecanismos de presentación. Por ejemplo, una interfaz de usuario gráfica.

En ingeniería del diseño existe la suposición implícita de que hasta el más fantasioso de los ensamblajes tiene funciones prácticas que satisfacer. Casos como la función de componentes físicos (ej. un pedal) en un artefacto diseñado (ej. una bicicleta) son discutidos y la descomposición funcional es un método popular para diseñar artefactos reales (ej. una planta eléctrica) y virtuales (ej. software) [2].

Aunque los diseñadores de sistemas de software desarrollaron el concepto de descomposición funcional y análisis estructural en 1960, la idea de usar el concepto funcional en la ingeniería de diseño fue primeramente mencionada por Herbert Simon en 1969; esta idea fue presentada más tarde por Freeman y Newell, quienes en 1971 desarrollaron un modelo para el razonamiento funcional en diseño, donde cada estructura (estructura que podría ser una entidad física que tiene atributos que le permiten proporcionar las funciones especificadas) se describe por las funciones que puede proporcionar, y para proporcionar cada una de éstas, las funciones que requiere. La idea fundamental consiste en encontrar una estructura final que pueda ser compuesta a partir de las estructuras disponibles y que satisfaga las propiedades deseadas [4].

Este primer modelo de Razonamiento Funcional, pionero en lo que a configuración se refiere, ha dado lugar a múltiples investigaciones y otras definiciones de configuración así como modelos de Razonamiento Funcional han ido apareciendo. En el año 1989 Mittal y Frayman definieron la configuración como un tipo especial de actividad de diseño en la cual el artefacto a diseñar es construido a partir de un conjunto de componentes predefinidos que solo pueden ser conectados de ciertas maneras [5], esta definición de configuración coincide exactamente con la clasificación de uno de los tipos de problemas que se presentan en el razonamiento funcional, específicamente los

problemas de selección. Es por esto que cuando resolvemos problemas de selección a través de un modelo de razonamiento funcional, pues estamos resolviendo problemas de configuración.

Desde el año 1971, en que Freeman y Newell propusieron el primer modelo de razonamiento funcional en diseño hasta la fecha, diferentes autores han combinado ideas en aras de lograr una propuesta que solucione este tipo de problemas de forma eficiente. En el año 2001, investigadores de la Universidad de Cambridge realizan un estudio en el cual se describen, además del modelo de 1971, los otros dos modelos que hasta ese momento se conocían, estos son: El modelo paradigma, propuesto por Yoshikawa [6, 7] y el modelo sistemático, propuesto por Pahl and Beitz [8], estos investigadores analizaron críticamente estos modelos existentes y propusieron uno nuevo, basándose en las dificultades encontradas [9].

En general, los supuestos y teorías detrás de las técnicas de razonamiento funcional no son usualmente claras y las propuestas de modelos computacionales para el diseño es un campo en desarrollo que no muestra resultados concretos prácticos [2].

En la actualidad puede constatarse que se han desarrollado diferentes modelos y técnicas para resolver problemas de razonamiento funcional siguiendo distintos criterios, pero no existe una concesión sobre que modelo es el idóneo ni cuales algoritmos los mejores para cada tipo de problema. Por lo anteriormente expuesto, se deriva el **problema científico** a resolver que se manifiesta en ausencia de métodos que resuelvan problemas de razonamiento funcional, específicamente problemas de configuración, de forma eficiente.

Para contribuir a la solución del problema científico antes plateado, se formuló la **hipótesis general de investigación** siguiente:

A partir del empleo de metaheurísticas es posible desarrollar métodos que son eficientes en la solución de problemas de configuración utilizando modelos de razonamiento funcional.

Esta hipótesis quedará validada si se comprueba que:

1. Es posible aplicar algoritmos heurísticos, específicamente Ascensión de Colinas, Algoritmos Genéticos y Optimización basada en colonias de hormigas para resolver problemas de razonamiento funcional.
2. La implementación de los algoritmos permite obtener resultados satisfactorios para diferentes bases de datos de componentes.

En conformidad con la hipótesis de investigación identificada, el **objetivo general** de la investigación consiste en implementar diferentes algoritmos heurísticos para la solución de

problemas de configuración usando un modelo de razonamiento funcional, con vistas a determinar cuán eficientes resultan a partir de la comparación de los resultados obtenidos de aplicarlos a múltiples bases de datos de componentes.

Este objetivo general fue desglosado en los **objetivos específicos** siguientes:

1. Analizar la aplicabilidad de métodos heurísticos, específicamente Hill-Climbing, Algoritmos Genéticos y Ant Colony System para resolver problemas de configuración.
2. Implementar los algoritmos seleccionados.
3. Realizar un estudio comparativo entre ellos como vía de comprobación y factibilidad de la investigación realizada, a partir de los resultados obtenidos usando distintas bases de datos de componentes.

La **novedad científica** principal que aporta esta investigación, radica en la solución dada al problema de configuración utilizando los métodos Ascensión de Colinas, Sistema de Colonias de hormigas y Algoritmos Genéticos.

El **valor teórico** de la investigación está directamente vinculado con su novedad científica.

El **valor práctico** se relaciona con la introducción de tres nuevos algoritmos que posibilitan resolver problemas de configuración partiendo de bases de datos de componentes, algoritmos que brindan la posibilidad de ser aplicados en la solución de problemas reales por el grupo de Inteligencia Artificial de nuestro centro.

Para el desarrollo de esta investigación se utilizaron métodos y técnicas de análisis y síntesis, análisis inductivo, herramientas matemáticas, procesamiento computacional de los resultados, sin excluir el análisis lógico, la analogía, la reflexión y otros procesos mentales que también son inherentes a toda actividad de investigación científica.

Para la presentación de esta investigación, esta Tesis de Maestría se estructuró de la forma siguiente. Una Introducción, donde en lo esencial se caracteriza la situación problémica y se fundamenta el problema científico a resolver, así como la estrategia general seguida para su solución como problema científico. Un Capítulo 1, que contiene el marco teórico-referencial que sustentó la investigación originaria. Un Capítulo 2, en el que se resumen y explican los algoritmos propuestos. Un Capítulo 3, donde se pone a prueba el desempeño de los algoritmos y se hace un estudio comparativo con los resultados obtenidos por cada uno de ellos. Un cuerpo de Conclusiones y Recomendaciones derivadas de la investigación realizada, la Bibliografía consultada y un anexo como complemento de los resultados expuestos.

Capítulo 1 Problemas de Razonamiento Funcional y Métodos Heurísticos

El estudio y la aplicación del razonamiento funcional en diferentes áreas han permitido establecer una clasificación de los tipos de problemas que pueden presentarse en 4 grupos, estos son [2]:

Problemas de Identificación: Dado un sistema, su función es explicada usando el conocimiento de la estructura y el comportamiento de sus componentes y su organización. Por ejemplo, cual es la función de un par de tijeras.

Problemas de Explicación: Consisten en explicar la presencia de un componente en un sistema en términos de su contribución a la función global del mismo.

Problemas de Selección: Dado un conjunto de componentes, seleccionar una combinación apropiada de ellos de forma que, si son usados juntos, puedan satisfacer los requerimientos del sistema.

Problemas de Verificación: Verificar cuando un elemento puede exhibir una función requerida en una situación dada.

En este capítulo se abordan en profundidad los problemas de selección, particularmente en ingeniería de diseño. Para esto se analizan los modelos existentes para enfrentar este tipo de problema y como elemento importante para la solución de los mismos son presentados algunos métodos heurísticos que se pueden emplear en el proceso de búsqueda. A partir de este análisis se presentan los problemas que se abordan en esta tesis.

1.1 Modelos de Razonamiento Funcional en ingeniería de diseño

En un estudio realizado en el 2001 por investigadores de la Universidad de Cambridge [9], se plantea que históricamente, además del modelo propuesto por Freeman y Newell en 1971, han existido otros dos modelos de Razonamiento Funcional influyentes en la ingeniería de diseño, el propuesto por Yoshikawa, conocido como el modelo paradigma, y el propuesto por Pahl y Beitz en 1992 llamado modelo sistemático, los autores de este estudio analizan críticamente los tres modelos y proponen uno nuevo en base a las dificultades encontradas.

El modelo original, que ha servido de guía y cuya forma de representación ha perdurado, plantea que una estructura se puede describir por las funciones que puede proporcionar, y para proporcionar cada una de estas, las funciones que requiere (Fig 1.1) [4].

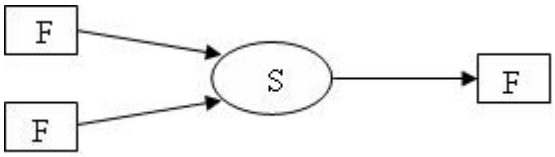


Fig. 1.1. Descripción funcional de una estructura S

La búsqueda de una solución a un problema dado, utilizando el modelo de Freeman y Newell, definido en términos de un conjunto de funciones deseadas, empezaría con una búsqueda entre las estructuras conocidas para encontrar aquellas que pueden proporcionar las funciones deseadas.

Estas estructuras escogidas, a su vez, requieren algunas otras funciones para poder proporcionar las funciones deseadas. Ahora, nuevas estructuras serían buscadas para proporcionar estas funciones requeridas que a su vez dan lugar a los nuevos requisitos funcionales. Este proceso continuará hasta que todas las funciones requeridas sean proveídas por algunas estructuras. Para esto existen ciertos postulados:

- Una conexión funcional puede aparecer entre dos estructuras si una ofrece una función requerida por la otra.
- Una estructura construida está formada por un conjunto de estructuras (sus partes) y un conjunto de conexiones funcionales entre ellas tal que:

    a) Las funciones ofrecidas por la estructura compuesta a su ambiente externo son aquellas ofrecidas por las partes las cuales no se consumen en las conexiones funcionales establecidas.

    b) Las funciones requeridas por la estructura compuesta desde su ambiente externo son aquellas requeridas por las partes las cuales no son satisfechas por las conexiones funcionales entre ellas.

Solamente aquellas funciones ofrecidas que no se consumen por conexiones funcionales y las necesidades no satisfechas por conexiones funcionales se mantienen en la interfaz funcional del objeto compuesto.

El objeto configurado satisface la especificación para el artefacto deseado aunque la interfaz funcional no sea exactamente la misma especificada para este. Un objeto compuesto el cual se presenta como una solución a la tarea de configuración tiene que ofrecer al menos todas aquellas funciones que debe ofrecer el artefacto deseado y tiene que necesitar no más que aquellas funciones que pueden ser suministradas al mismo[10].

En el modelo paradigma, un problema de diseño es expresado en términos de un conjunto de requerimientos funcionales y las soluciones en términos de un conjunto de atributos que posibilitan

encontrar funciones específicas. El diseñador propone una solución que parece cumplir el mejor de los requisitos, luego la evalúa identificando los componentes inapropiados o erróneos en la solución que la hacen no satisfactoria para la especificación inicial, y reformula lo que todavía necesita satisfacer al establecer la diferencia entre la especificación original y lo que provee la solución provisional propuesta. Se buscan entonces nuevos componentes provisionales que sean capaces de reducir esta diferencia, reemplazando los inapropiados con estos nuevos, es decir, formando una segunda y mejor solución provisional. Este proceso continúa hasta que se encuentra una solución satisfactoria [6, 7, 9].

En el modelo propuesto por Pahl y Beitz en 1992, un problema de diseño se divide buscando módulos funcionales por separado, cada módulo puede considerarse entonces independientemente, con la interacción entre ellos mantenida al mínimo. La mayor ventaja de este modelo es la simplificación del proceso de diseño subsiguiente para los módulos individuales [8].

Es decir, que el problema debe ser expresado como una función de solución neutral o conjunto de funciones de soluciones neutrales; es decir, se modela una o varias funciones globales que son progresivamente expandidas en combinaciones de sub-funciones; dichas combinaciones son llamadas función-estructura. Este proceso de simplificación es continuo hasta que las sub-funciones (funciones por las que una función-estructura está compuesta) son suficientemente simples. Es posible generar otras variantes de una función-estructura a través de recombinaciones de sus sub-funciones.

Luego, la función-estructura óptima es escogida y son buscadas las posibles alternativas de solución para cada sub-función, estas soluciones son combinadas en conceptos de solución alternativos, que son luego evaluados y el más prometedor escogido.

Los tres modelos anteriormente explicados tienen deficiencias. Esencialmente, el modelo de Freeman y Newell representa un proceso de avance a través de los diferentes niveles de detalle del diseño, pero no da ningún tipo de ayuda para superar la frecuente dificultad de encontrarse en un punto donde ninguna estructura propuesta satisface completamente los requerimientos funcionales [9].

La deficiencia fundamental en el modelo paradigma es que al proponerse una solución al problema inicial, luego se identifican los componentes erróneos para reformular dicho problema como la diferencia existente las especificaciones iniciales y la solución propuesta, esta reformulación debería ser hecha teniendo en cuenta la solución propuesta ya una vez eliminados los componentes erróneos.

En el modelo sistemático la mayor desventaja está en que disminuyendo el ámbito de cooperación funcional, se puede provocar una mayor complejidad global del producto o componente a diseñar.

Además, se presentan preguntas como:

- ¿Cuándo se detiene el proceso de elaboración de funciones-estructuras?
- ¿Cómo se garantiza que este proceso acerca el problema reformulado a su posible solución?

En base a todo lo explicado se propone en el 2001 un nuevo modelo el cual [9]:

- Representa las estructuras usando la representación propuesta por Freeman y Newell en 1971
- Opera con un conjunto conocido de estructuras o componentes
- Garantiza la obtención de una solución a un problema determinado si existe conexión funcional entre las estructuras conocidas
- Utiliza un esquema para la construcción de las soluciones basado en el modelo de Freeman y Newell

El modelo funciona de la forma siguiente:

Dado: Un conjunto de estructuras, cada una de las cuales provee ciertas funciones y necesita otras para poder proveer estas (exactamente como en el modelo de Freeman y Newell) y un problema de diseño o configuración, definido en términos de un conjunto de funciones.

Se requiere: Combinaciones de estructuras, compuestas por las estructuras mencionadas anteriormente, que sean capaces de solucionar el problema de configuración dado.

Esquema:

Se plantea el problema inicial y se van buscando soluciones parciales, es decir, el problema no necesita ser resuelto como un todo, sino que puede ser dividido en partes. En cada paso se incorpora una nueva estructura que aporta a lo que será la solución final; la redefinición del problema se realiza en base al problema existente en el paso anterior, la contribución de la nueva estructura seleccionada y las funciones que la misma requiere.

Debido a las fortalezas que presenta con respecto al resto, éste fue el modelo escogido para aplicar en la búsqueda de soluciones.

Antes de poner un ejemplo es importante destacar que existen dos formas de enfrentar estos problemas de búsqueda, hacia atrás (*backward*) o hacia adelante (*forward*). La búsqueda hacia atrás comienza con las funciones deseadas y trabaja, como su nombre lo indica, hacia atrás en busca de las estructuras disponibles que puedan ir satisfaciendo la necesidad existente, mientras que hacia adelante se parte de las estructuras que pueden consumir hasta que aparezca una que ofrezca las funciones requeridas.

Un ejemplo podría ser el siguiente [11], dados los componentes:

Componente	**Consume**	**Ofrece**
A	f1, f3	f2, f5
B	f4	f3
C	f1, f5	f6

Se desea una configuración que ofrezca la función f6 consumiendo como máximo 2f1 y f4.

El proceso de razonamiento funcional en dirección *forward* es:

Disponible	Estructura	Necesidad pendiente
2f1, f4	∅	f6
2f1, f3	B	f6
f1, f2, f5	B, A	f6
f2	B, A, C	--

En este diseño queda disponible la función f2.

El proceso de razonamiento funcional en dirección *backward* es:

Disponible	Estructura	Necesidad pendiente
2f1, f4	∅	f6
f1, f4	C	f5
f4, f2	C, A	f3
f2	C, A, B	--

En este diseño también queda disponible la función f2.

En este trabajo se utiliza la búsqueda hacia atrás, y será esta la forma de búsqueda que se abordará con detalle y se utilizará en un próximo ejemplo, aunque ambos métodos tienen un carácter combinatorio de búsqueda heurística. En cada estado del diseño está disponible un conjunto de posibles acciones que permiten avanzar en el mismo, una de las cuales debe ser seleccionada, conduciendo de esta forma a una nueva situación de diseño parcial. Al igual que en otros problemas similares, el conjunto de alternativas es generalmente tan grande que, la búsqueda por fuerza bruta (por ejemplo primero a lo ancho, *breadth first*) posiblemente no pueda tener éxito [4].

Cuando se desarrolla la búsqueda hacia atrás, al escoger una estructura debe tenerse en cuenta que todo lo que ofrece que no está entre las funciones deseadas pasa a ser consumo disponible, y aquellas funciones que consume que no estaban disponibles en el consumo en ese momento, pasan a ser parte de las próximas funciones a buscar.

En este caso se muestra un ejemplo donde se hace uso de una base de datos de estructuras mecánicas que cuenta con lo siguientes componentes (ver Fig. 1.2):

- una estructura compuesta, consistiendo en dos engranajes de estímulo y un eje intermedio que los conecta. Su función es transformar una fuerza de entrada, transferida adentro a través de una interfaz con engranaje de dientes o dentada, en una fuerza de salida, transferida también a través de una interfaz dentada. La entrada y la salida están en los planos paralelos;
- un engranaje de estímulo, que se puede utilizar para tomar una fuerza de entrada a través de un interfaz dentada, y transformarla en una fuerza de torsión que es perpendicular a la fuerza, o viceversa;
- un eje que es capaz de transmitir una fuerza de torsión a través de su longitud;
- una leva, pieza que gira alrededor de un punto que no es su centro, transformando el movimiento circular en uno rectilíneo (árbol de levas); y
- una cuña, que toma una fuerza de entrada a través de una interfaz de contacto y entrega una fuerza de salida a un ángulo especificado de la entrada, a través de otra interfaz del contacto.

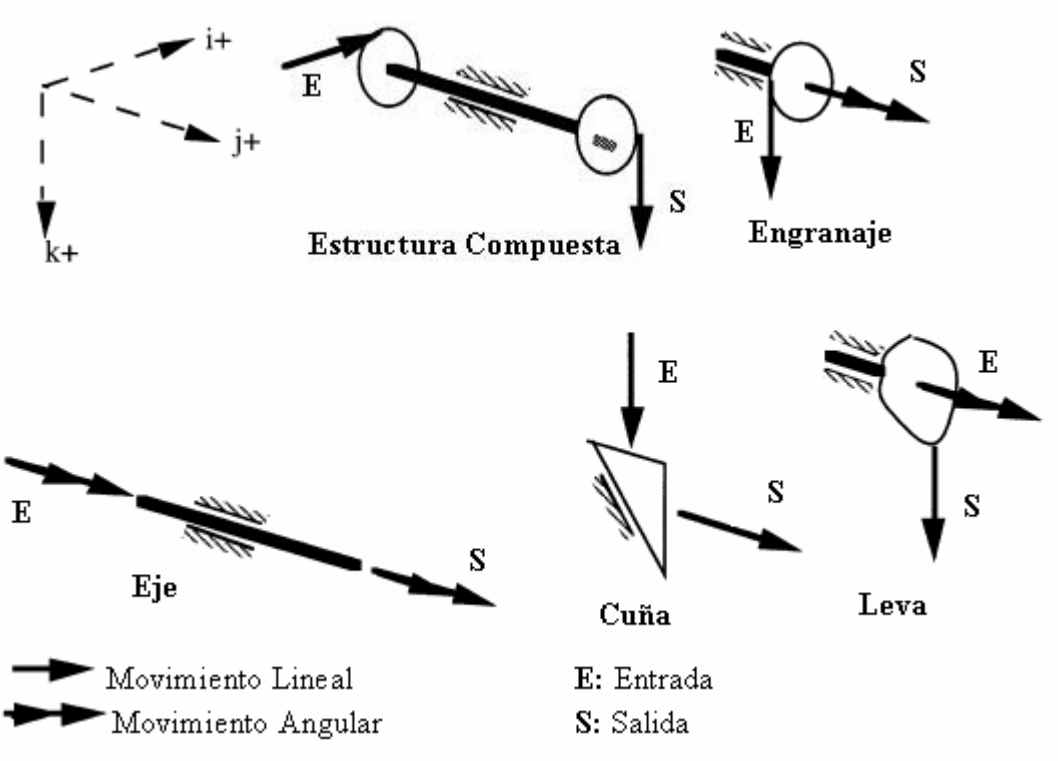
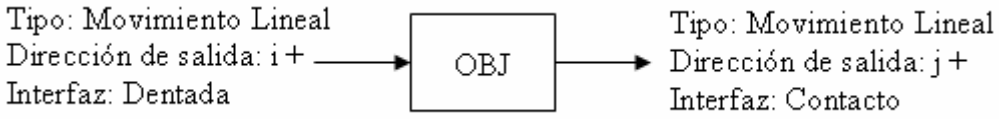


Fig. 1.2. Base de datos de estructuras mecánicas

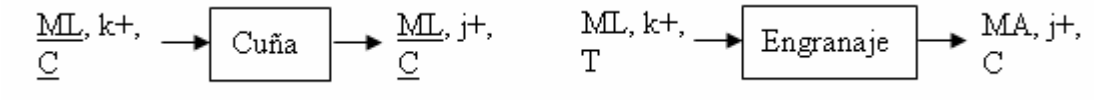
El problema de diseño consiste en transformar una fuerza de entrada en una dirección positiva i, disponible a través de una interfaz dentada, en una fuerza de salida en una dirección positiva j, para ser transferido a través de una interfaz de contacto (ésta puede ser, entre otras, parte de una función de fijación de una puerta).

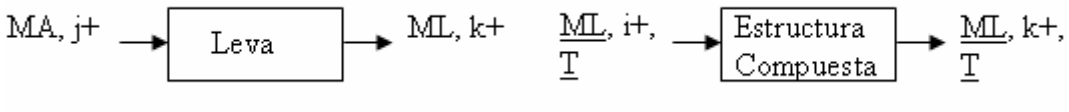


Son utilizadas las siguientes siglas para representar cada uno de los componentes con sus respectivas funciones de forma más clara:

ML: Movimiento Lineal **D:** Interfaz Dentada

MA: Movimiento Angular **C:** Interfaz de Contacto

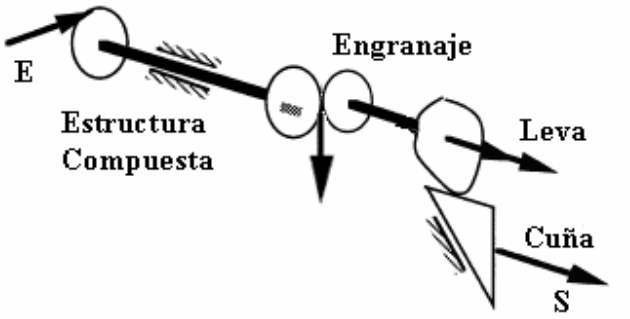




Se encuentran subrayadas aquellas funciones que se anulan, es decir, que son consumidas y ofrecidas por la misma estructura. Desarrollando la búsqueda de una solución hacia atrás (*backward*), el proceso es el siguiente:

Consumo Disponible	**Estructura**	**Necesidad pendiente**
ML, i+, D	∅	ML, j+, C
i+, D	Cuña	k+, C
i+, D, ML	Cuña, Leva	MA, j+, C
i+	Cuña, Leva, Engranaje	k+
∅	Cuña, Leva, Engranaje, Estructura Compuesta	∅

Es decir, que una solución para el problema planteado es:



Dada la complejidad computacional del proceso de construcción de configuraciones, el cual es dependiente de la cantidad de componentes, este cae en la categoría de los problemas de optimización, pues se desea minimizar la cantidad de componentes o el valor total de la estructura resultante. Para resolverlo se hace necesario el empleo de métodos heurísticos en la solución de este problema.

1.2 Optimización y métodos heurísticos

Optimización en el contexto científico es el proceso de tratar de encontrar la mejor solución posible para un determinado problema. En un problema de optimización existen diferentes soluciones factibles y un criterio para discriminar entre ellas; de forma más precisa, estos problemas se pueden expresar como: encontrar el valor de ciertas variables de decisión para los que una determinada

función objetivo alcanza su valor máximo o mínimo. El valor de las variables, en ocasiones, está sujeto a restricciones.

Es posible identificar múltiples problemas de optimización, tanto en la industria como en la ciencia. Desde los clásicos problemas de organización de la producción y diseño de redes de telecomunicación hasta los más actuales en ingeniería de software, existe una infinidad de problemas teóricos y prácticos que involucran a la optimización.

Algunas clases de problemas de optimización son relativamente fáciles de resolver, sin embargo, la mayor parte de los que se encuentran en la práctica pueden ser clasificados como muy difíciles, entendiéndose como problema de optimización difícil aquel para el cual no se puede garantizar encontrar la mejor solución posible en un tiempo razonable.

La existencia de gran cantidad y variedad de problemas difíciles, que aparecen en la práctica y que necesitan ser resueltos de forma eficiente, impulsó el desarrollo de procedimientos eficientes para encontrar buenas soluciones aunque no fueran óptimas. Estos métodos en los que la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan heurísticos o aproximados.

El concepto de heurística ofrecido por Feigenbaum y Feldman [12]es el siguiente:

"Una heurística es un tipo de estrategia que limita en forma drástica la búsqueda de soluciones. La heurística no garantiza soluciones óptimas; de hecho, no garantiza el que haya una solución; todo lo que se puede decir para que una heurística sea útil es que ofrece soluciones que son suficientemente buenas la mayoría de las veces"

En esencia una heurística es simplemente un conjunto de reglas que evalúan la posibilidad de que una búsqueda va en la dirección correcta. Generalmente los métodos de búsqueda heurística se basan en maximizar o minimizar algunos aspectos del problema; en [13] se recogen más de dos definiciones diferentes de algoritmo heurístico, entre las que se ha querido destacar la siguiente:

Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución.

En contraposición a los métodos exactos que proporcionan una solución óptima del problema, los métodos heurísticos se limitan a proporcionar una buena solución no necesariamente óptima. Lógicamente, el tiempo invertido por un método exacto para encontrar la solución óptima de un problema difícil, si es que existe tal método, es de un orden de magnitud muy superior al del heurístico (pudiendo llegar a ser tan grande en muchos casos, que sea inaplicable).

En los últimos años ha habido un crecimiento espectacular en el desarrollo de procedimientos heurísticos para resolver problemas de optimización. Este hecho queda claramente reflejado en el gran número de artículos publicados en revistas especializadas. En 1995 se edita el primer número de la revista Journal of Heuristics dedicada íntegramente a la difusión de los procedimientos heurísticos.

Aunque solamente se ha mencionado la problemática de la resolución de un problema difícil, existen otras razones para utilizar métodos heurísticos, entre las que se pueden destacar:

- El problema es de una naturaleza tal que no se conoce ningún método exacto para su resolución.
- Aunque existe un método exacto para resolver el problema, su uso es computacionalmente muy costoso.
- El método heurístico es más flexible que un método exacto, permitiendo, por ejemplo, la incorporación de condiciones de difícil modelación.
- El método heurístico se utiliza como parte de un procedimiento global que garantiza el óptimo de un problema; existiendo dos posibilidades:
    - ✓ El método heurístico proporciona una buena solución inicial de partida.
    - ✓ El método heurístico participa en un paso intermedio del procedimiento.

Al abordar el estudio de los algoritmos heurísticos se puede comprobar que dependen en gran medida del problema concreto para el que se han diseñado. En otros métodos de resolución de propósito general, como pueden ser los algoritmos exactos de ramificación y acotación, existe un procedimiento conciso y preestablecido, independiente en gran medida del problema abordado. En los métodos heurísticos esto no es así. Las técnicas e ideas aplicadas a la resolución de un problema son específicas de este y aunque, en general, pueden ser trasladadas a otros problemas, han de particularizarse en cada caso. Así pues, es necesario referirse a un problema concreto para estudiar con detalle los procedimientos heurísticos.

En los últimos años han aparecido una serie de métodos bajo el nombre de metaheurísticos con el propósito de obtener mejores resultados que los alcanzados por los heurísticos tradicionales, este término fue introducido en [14] por Fred Glover. En este trabajo se hará referencia a los métodos clásicos utilizando el término heurísticos, reservando el de metaheurísticos para los más recientes y complejos. En algunos textos podemos encontrar la expresión “heurísticos modernos” refiriéndose a los metaheurísticos [15]. Los profesores Osman y Kelly en [16] introducen la siguiente definición:

Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Los Metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.

Los procedimientos metaheurísticos se sitúan conceptualmente "por encima" de los heurísticos en el sentido que guían el diseño de éstos. Así, al resolver un problema de optimización, puede escogerse cualquiera de estos métodos para diseñar un algoritmo específico que lo resuelva aproximadamente.

En estos momentos existe un gran desarrollo y crecimiento de estos métodos los cuales han probado su eficiencia sobre una colección significativa de problemas; entre los que se destacan la Búsqueda Tabú [17], el Recocido Simulado [18], los Algoritmos Genéticos [19], la Optimización Basada en Enjambre de Partículas [20] (*Particle Swarm Optimization, PSO*) y la Optimización Basada en Colonias de Hormigas (*Ant Colony Optimization, ACO*) [21, 22], entre otros. Estas últimas dos Metaheurísticas caen en la categoría de algoritmos bioinspirados o de vida artificial e inteligencia colectiva, ya que la potencialidad de estos modelos para resolver problemas está dada por la cooperación entre individuos de una forma directa o indirecta.

1.2.1 Ascensión de Colinas (*Hill-Climbing*)

Para entender el significado del nombre del algoritmo, se puede imaginar el espacio de todas las posibles soluciones como un paisaje del entorno, donde un conjunto de coordenadas representa una solución particular. Las soluciones son mejores a mayor altitud formando colinas o picos y peores a menor altitud formando valles. Se comienza en un punto dado de ese paisaje haciendo movimientos ascendentes (*climbing*), como un bucle que se mueve en dirección del valor creciente y termina cuando alcanza un pico en el que ningún vecino tiene valor mejor [23].

El algoritmo comienza con una solución generalmente aleatoria, que de acuerdo con cierto mecanismo va modificándose. Si se encuentra una solución que aumente el valor de la función objetivo nos quedamos con ella, en caso contrario se desecha, manteniendo la que se tenía. Este proceso continúa hasta que se encuentre la solución que maximiza la función objetivo o hasta que cumpla un número máximo de iteraciones, devolviendo como resultado la solución encontrada.

Con este método la estrategia es repetidamente expandir un nodo, inspeccionar sus sucesores recién generados, y seleccionar y expandir el mejor entre los sucesores sin mantener referencias a los padres. Existen distintos criterios para seleccionar el próximo nodo, pues el criterio que se siga, que no es más que el punto por donde se continuará la búsqueda, también influye en la calidad del algoritmo [24]. Las alternativas son las siguientes:

- Escalada de primera opción: Escoge el primero que mejore el estado actual.
- Escalada estocástica: Se escoge aleatoriamente entre los movimientos ascendentes.
- Escalada por máxima pendiente: Se selecciona el mejor movimiento (no el primero de ellos) que suponga mejora respecto al estado actual.

Consecuencias:

- Escoger el primero que mejore la solución actual permite cálculos más rápidos pues no es necesario analizar todos los vecinos.
- Escoger el mejor movimiento o uno aleatorio entre los que mejoran el estado actual requieren calcular todos los vecinos antes de determinar cómo continuar la búsqueda y, por lo tanto, implican mayor cantidad de cálculos.
- En general, escoger el mejor movimiento entrega mejores resultados, aún cuando una selección aleatoria también puede hacerlo.

Este método es una variación de la búsqueda primero en profundidad en la cual se usa la función heurística para estimular la distancia entre el nodo actual y el nodo objetivo.

El algoritmo ascensión de colinas es típicamente local, ya que la decisión para continuar la búsqueda se toma mirando únicamente a las consecuencias inmediatas de sus opciones. Puede que nunca llegue a encontrar una solución, si queda atrapado en estados que no son el objetivo y desde donde no se puede hallar mejores estados, por ejemplo:

- Máximo local: Ningún vecino tiene mejor coste.
- Meseta: Todos los vecinos son iguales.
- Cresta: La pendiente de la función sube y baja (efecto escalón).

Fig. T.

1.2.2 Algoritmos Genéticos (AG)

Los Algoritmos Genéticos surgen como herramientas para la solución de complejos problemas de búsqueda y optimización, producto del análisis de los sistemas adaptativos en la naturaleza y como resultado de abstraer la esencia de su funcionamiento [25].

De manera muy general, se puede decir que en la evolución de los seres vivos el problema al que cada individuo se enfrenta es la supervivencia. Para ello, cuenta con las habilidades innatas provistas en su material genético. A nivel de los genes, el problema es el de buscar aquellas adaptaciones beneficiosas en un medio hostil y cambiante [26].

Desde finales de los años 50 y principios de los 60 fueron desarrollados por algunos biólogos trabajos para simular sistemas genéticos en una computadora. A pesar de estos trabajos, se reconoce al profesor John Holland de la Universidad de Michigan como el creador de los AGs con su trabajo sobre teoría de los sistemas adaptativos, en el año 62. Fue él quien primero adaptó la idea de la genética a sistemas artificiales y posteriormente, en el año 75, introdujo mejoras importantes en los AGs como es el escalado y publicó un libro sobre el tema que es material de referencia clásico [27].

Los Algoritmos Genéticos son un ejemplo de método que explota la búsqueda aleatoria "guiada" que ha ganado popularidad en los últimos años debido a la posibilidad de aplicarlos en una gran gama de campos y a las pocas exigencias que impone al problema. Se han aplicado satisfactoriamente en los más disímiles problemas de optimización, como planificación, control adaptativo, juegos, modelación cognitiva, problemas de transportación, problema del viajero vendedor, problemas de control óptimo, optimización en Base de Datos: optimización de solicitudes, etc, [28-31].

1.2.2.1 Terminología usada

Los AGs son técnicas de búsqueda basadas en los principios de la selección natural de Darwin, es por esto que no se puede hablar de ellos sin mencionar los conceptos biológicos con los que tienen relación. Las analogías entre los AGs y selección natural son claras:

Genotipo: Juega un papel fundamental en el proceso evolutivo, constituye el material hereditario presente en todos los organismos vivos, no es observable a simple vista; en términos del AG el genotipo es la información genética codificada del cromosoma.

Fenotipo: Es el paquete genético tal y como interactúa con el medio exterior, es en cualquier organismo el conjunto de caracteres que se pueden observar a simple vista; en los AG constituyen los aspectos del cromosoma decodificados, o sea los valores de la función de evaluación.

Genes: Constituyen cadenas de ADN que portan los caracteres hereditarios. En un AG son representados como las variables del problema a resolver, los componentes de los cromosomas.

Cromosoma: Es el portador de la información genética, constituye el conjunto de genes; en nuestro caso es una posible solución del espacio de búsqueda.

Población: Conjunto de individuos o cromosomas; en los AG constituyen una muestra aleatoria del espacio o un conjunto de soluciones alternativas.

1.2.2.2 Componentes de un Algoritmo Genético

Los Algoritmos Genéticos están basados en integrar e implementar eficientemente dos ideas fundamentales [32]:

- Las representaciones simples, como cadenas binarias, de las soluciones del problema.
- La realización de transformaciones simples para modificar y mejorar estas representaciones.

Para llevar a la práctica el esquema anterior y concretarlo en un algoritmo, hay que especificar los siguientes elementos:

- Una representación cromosómica;
- una población inicial;
- una medida de evaluación *(aptitud o fitness);*
- un criterio de selección / eliminación de cromosomas;
- una o varias operaciones de cruzamiento;

- una o varias operaciones de mutación

Estos tres últimos son los llamados operadores genéticos clásicos, aunque otros operadores también pueden ser usados [33].

En los primeros trabajos [25] las soluciones se representaban por cadenas binarias, es decir, listas de 1s y 0s. Este tipo de representación ha sido ampliamente utilizada incluso en problemas en donde no es muy natural.

La población inicial suele ser generada en forma aleatoria, sin embargo, últimamente se están utilizando métodos heurísticos para generar soluciones iniciales de buena calidad.

Respecto a la evaluación de los cromosomas, se suele utilizar la calidad como medida de la bondad según el valor de la función objetivo en el que se puede añadir un factor de penalización para controlar la infactibilidad.

La selección es el proceso que se encarga de la elección de los mejores individuos para que participen en el cruce. La forma de escoger los padres viene dada habitualmente mediante probabilidades calculadas según su aptitud. Existen distintas variantes para realizar la selección, por ejemplo, selección por torneo, basada en el rango, elitismo, entre otras, pero uno de los procedimientos más utilizado es el denominado "Método de la ruleta", en el cual cada individuo tiene una sección de una ruleta circular cuyo tamaño es directamente proporcional a su calidad (medida de evaluación).

Es a esta "preselección" obtenida por selección a la que se aplica el resto de los operadores siendo el resultado de este proceso la nueva población.

En la etapa de cruzamiento los individuos seleccionados se "aparean" para intercambiar parte de su información genética. A partir de este intercambio se obtienen nuevos individuos que tienen una cadena que posee características de ambos padres y que serán los componentes de la nueva generación.

Los operadores de cruzamiento más utilizados son:

- De un punto: Se elige aleatoriamente un punto de ruptura en los padres y se intercambian sus bits.
- De dos puntos: Se eligen dos puntos de ruptura al azar para intercambiar.
- Uniforme: Es el intercambio de bits entre las cadenas, en vez de segmentos como los casos anteriores. En cada posición de la cadena los bits son probabilísticamente intercambiados con una probabilidad fija.

Para llevar el control se trabaja con un parámetro denominado Probabilidad de Cruzamiento, es decir, que para que el AG realice el cruzamiento tiene que generar un número aleatorio para determinar si ocurrirá o no el mismo en función de la Probabilidad de Cruzamiento definida.

Por último nos referiremos a la mutación, que constituye un proceso similar al biológico en el cual un gen de un individuo cambia su valor a otro valor posible. Se puede notar que el papel que juega la mutación es el de introducir un factor de diversificación ya que, en ocasiones, la convergencia del procedimiento a buenas soluciones puede ser prematura y quedarse atrapado en óptimos locales.

Puede ser que ninguno de los individuos de la población tenga una característica (y que hasta incluso ninguno de los anteriores tuvo) y que esa característica es el detalle que necesitan para lograr una mejoría importante, entonces esto puede obtenerse también a través de la mutación.

El proceso es muy simple y solo consiste en determinar cual es el gen que mutará el organismo. La selección de este gen se realiza de forma aleatoria. Una vez que se obtiene solo es necesario cambiar el valor que está en esa posición de la cadena por su opuesto.

Para el AG es importante tener control de la mutación ya que estos cambios pueden tener tanto un efecto conveniente como perjudicial.

Para esto se utiliza un parámetro llamado Probabilidad de Mutación, el cual servirá para determinar si un individuo mutará o no. Como se ve aquí tampoco se trabaja de forma determinística sino a través de las probabilidades; la probabilidad de que este proceso ocurra debe ser muy baja asegurando de esta forma que no se convierta en sobreactivo, lo cual daría al traste con el algoritmo destruyendo los esquemas y con esto las ventajas logradas con el cruzamiento.

1.2.2.3 Funcionamiento de un Algoritmo Genético

El modo de trabajo de un AG puede resumirse en el esquema siguiente, el cual se ejecuta durante un número fijo de generaciones o hasta que se satisface algún criterio de parada:

- El AG simple genera aleatoriamente una población de n estructuras (cadenas, cromosomas o individuos).
- Se evalúan los individuos de acuerdo a la función.
- Sobre la población actúan los operadores explicados anteriormente ejerciendo transformaciones en la misma. Una vez completada la acción de los tres operadores se dice que ha transcurrido un ciclo generacional.
- Se actualizan los individuos de la próxima generación a través de la comparación de poblaciones y se evalúa la población que persistirá.

- Luego se repiten los dos pasos anteriores mientras no se garantice el criterio de parada del AG.

Existen varios criterios de comparación de poblaciones, de los cuales, la comparación por el mejor y la comparación por el promedio son los más empleados. Estas sugieren que al comparar dos poblaciones, entiéndase la población actual y la anterior, si la población actual es peor (según el criterio tomado) que su progenitora, entonces se actualiza el algoritmo con la mejor de ambas (anterior), y se sigue iterando de forma usual. Es necesario destacar que el criterio escogido y la forma de implementar la comparación influirán en la determinación del criterio de parada del algoritmo.

1.2.3 Optimización basada en Colonias de Hormigas

La Vida Artificial [34-36] o algoritmos bioinspirados constituye una disciplina que estudia la vida natural, recreando los fenómenos biológicos en ordenadores y otros medios artificiales. Complementa el estudio teórico de la biología pero en lugar de tomar organismos aislados y analizar su comportamiento, lo que se intenta es colocar juntos organismos que actúan como los seres vivos. Esta técnica se puede encontrar en aplicaciones prácticas como en el diseño de ordenadores, robots móviles, medicina, nanotecnología, etc.
Una de las más recientes técnicas de Inteligencia Artificial en el área de Vida Artificial es la llamada Optimización Basada en Colonias de Hormigas (*Ant Colony Optimization, ACO*) [21, 22], metaheurística inspirada en el comportamiento (a nivel de especie) de colonias de hormigas que optimizan el camino para llegar desde el nido hasta su fuente de alimento, utilizando para éste fin, comunicación indirecta por medio de una sustancia química denominada feromona y su capacidad para adaptarse a cambios de ambiente.

Esta metaheurística desde su surgimiento ha representado una buena herramienta para solucionar problemas de optimización combinatoria, ofreciendo las técnicas algorítmicas más exitosas y ampliamente reconocidas basadas en el comportamiento de las hormigas, siendo la cooperación un componente clave en sus algoritmos.

Una hormiga artificial en ACO es un procedimiento constructivo estocástico que construye incrementalmente una solución agregando oportunamente componentes a la solución parcial en construcción. Las diferencias de las hormigas artificiales con respecto a las hormigas naturales son: (i) Tienen alguna memoria; (ii) No son completamente ciegas; y, (iii) Viven en un ambiente donde el tiempo es discreto.

1.2.3.1 Modo de funcionamiento y estructura genérica de la Metaheurística ACO

El modo de operación básico de un algoritmo de ACO es como sigue: las *m* hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los estados adyacentes del problema (que puede representarse en forma de grafo con pesos). Este movimiento se realiza siguiendo una regla de transición que está basada en la información local disponible, es decir, cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociados dos tipos de información que guían el movimiento de la hormiga:

- Información heurística: que mide la preferencia heurística de moverse desde el nodo r hasta el nodo s, o sea, de recorrer la arista a_{rs}. Se nota por η_{rs}. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- Información de los rastros de feromona artificiales: que mide la "deseabilidad aprendida" del movimiento de r a s. Imita a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas. Se denota por τ_{rs}.

Al moverse por el representativo del problema, las hormigas construyen incrementalmente soluciones. Opcionalmente, las hormigas pueden depositar feromona cada vez que crucen un arco en el proceso de construcción de la solución (*actualización en línea paso a paso de los rastros de feromona)*.

Una vez que cada hormiga ha generado una solución se evalúa ésta y puede depositar una cantidad de feromona que está en función de la calidad de su solución (*actualización en línea de los rastros de feromona*). Esta información guiará la búsqueda de las otras hormigas de la colonia en el futuro.

Es bueno aclarar que la estructura que almacena la feromona esta en dependencia del tipo del problema a resolver. Para problemas de secuenciación o asignación, la feromona es asociada a los arcos de grafo representativo, por lo que se habla de una matriz de feromona, como se presenta en la mayoría de los problemas. Para problemas donde el orden de los elementos de la solución no es importante se asocia la feromona a los nodos del grafo, por lo que se utiliza como estructura de almacenamiento un vector de feromona. En lo adelante, para explicar el funcionamiento de la metaheurística ACO y el comportamiento de sus algoritmos se utilizará el término "matriz de feromona", sin que se refiera a un tipo de problema en específico.

Los valores iniciales de feromona pueden ser de la siguiente forma:

- Inicialización aleatoria: donde cada elemento de la matriz es un elemento escogido aleatoriamente con distribución uniforme en el intervalo (0,1).

- Con un valor constante: cada elemento de la matriz es inicializado con el mismo valor, este valor debe de ser en el intervalo (0,1).
- Inicialización dependiendo de una solución (S) calculada por algún otro algoritmo con menor costo computacional; donde cada elemento de la matriz es inicializado con el valor 1/S.

Además, el modo de operación genérico de un algoritmo de ACO incluye dos procedimientos adicionales, la evaporación de los rastros de feromona y las acciones del demonio. La evaporación de feromona la lleva a cabo el entorno y se usa como un mecanismo que evita el estancamiento en la búsqueda y permite que las hormigas busquen y exploren nuevas regiones del espacio. Las acciones del demonio son acciones opcionales -que no tienen un contrapunto natural- para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen. Entre las capacidades adicionales que desarrolla las acciones del demonio están por ejemplo: observar la calidad de todas las soluciones generadas y depositar una nueva cantidad de feromona adicional sólo en las transiciones/componentes asociadas a algunas soluciones, o aplicar un procedimiento de búsqueda local a las soluciones generadas por las hormigas antes de actualizar los rastros de feromona. En ambos casos, el demonio reemplaza la actualización en línea *a posteriori* de feromona y el proceso pasa a llamarse *actualización fuera de línea de rastros de feromona*.

La estructura de un algoritmo de ACO genérico es como sigue [37, 38]

```
Procedimiento Metaheurística ACO;
  Actividades Programadas
       Construir Soluciones de las Hormigas
   Actualizar Feromona
         Evaporación de la feromona
        Acciones del Demonio       (opcional)
  Fin de las Actividades Fijas
Fin del procedimiento
```

Este procedimiento se anida en el siguiente procedimiento iterativo:

```
P1: Inicializar los valores de feromona
       Nciclo = 1
P2: Repetir
       Procedimiento  Metaheurística ACO
       Nciclo = Nciclo + 1
    Hasta que: criterio de parada
```

Los criterios de parada de este proceso iterativo pueden ser, entre otros:

- Se alcanza un número máximo de ciclos.
- Se alcanza una solución con la calidad deseada.
- Convergencia a la misma solución (estancamiento o stagnation).
- Se alcanza un tiempo límite de procesamiento.

1.2.3.2 Algoritmos de la Metaheurística ACO

Dentro de la metaheurística ACO pueden encontrarse una serie de algoritmos, por ejemplo, Sistema de Hormigas (*Ant System, AS*) [39], Sistema de Colonias de Hormigas (*Ant Colony System, ACS*) [40], el Sistema de Hormigas Max-Min (*Max-Min Ant System*) [41], el Sistema de Hormigas con ordenación (*Rank-Based Ant System*) [42], Hormigas Multitipo (*Multi type Ants*) y el Sistema de la Mejor-Peor Hormiga (*Best-Worst Ant System*) [43] , cada uno con sus características particulares.

Un gran número de investigadores, interesados por la originalidad y el rendimiento del modelo ACO, han aplicado la metaheurística con excelentes resultados a problemas tan diversos como:

- El paradigma del Viajante de Comercio (*Travelling Salesman Problem*) [39];
- El problema del ordenamiento secuencial (*Sequential Ordering Problem*) [44];
- El problema de ruteo de vehículos (*Vehicle Routing Problem*) [45];
- El problema de asignación cuadrática (*Quadratic Assignment Problem*) [46];
- Redes de telecomunicaciones (*Telecommunications Networks*) [47].

En el contexto de este trabajo se detallan el Sistema de Hormigas y el Sistema de Colonias de Hormigas.

Sistema de Hormigas

El Sistema de Hormigas (*Ant System*, *AS*) [39] fue desarrollado por Dorigo, Maniezzo y Colorni en 1991 y constituyó el primer algoritmo de ACO. Inicialmente, se presentaron 3 variantes distintas: Hormiga-Densidad (Ant-Density), Hormiga-Cantidad (Ant-Quantity) y Hormiga-Ciclo (Ant-Cycle), que se diferenciaban en la manera en que se actualizaban los rastros de feromona. En los dos primeros, las hormigas depositaban feromona mientras construían sus soluciones (es decir, aplicaban una actualización en-línea paso a paso de feromona), con la diferencia de que la cantidad de

feromona depositada en el algoritmo hormiga-densidad es constante, mientras que la depositada en hormiga-cantidad dependía directamente de la deseabilidad heurística de la transición η_{rs}. Por último, en hormiga-ciclo, la deposición de feromona se lleva a cabo una vez que la solución está completa (actualización en línea a posteriori de feromona). Esta última variante era la que obtenía mejores resultados y es por tanto la que se conoce como AS en la literatura.

El AS se caracteriza por el hecho de que la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones, y se lleva a cabo como sigue: primero, todos los rastros de feromona se reducen en un factor constante, implementándose de esta manera la evaporación de feromona según la expresión 1.1:

$$\tau_{ij} \leftarrow (1-p)*\tau_{ij},\ p \in (0,1] \tag{1.1}$$

donde p se conoce como constante de evaporación y es la encargada de reducir los rastros de feromona para evitar el estancamiento de las soluciones y τ_{ij} es la cantidad de feromona asociada al arco *(i, j)*.

A continuación, cada hormiga de la colonia deposita una cantidad de feromona que está dada en función de la calidad de su solución, según la expresión 1.2:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta t^k \quad \forall a_{ij} \in S^k \tag{1.2}$$

donde $\Delta t^k = f(C(S^k))$, es decir que la cantidad de feromona a depositar en cada arco (a_{ij}) que pertenece a la solución (S^k) de la hormiga k depende totalmente de la calidad ($C(S^k)$) de la solución encontrada por dicha hormiga.

Las soluciones en el AS se construyen como sigue. En cada paso de construcción, una hormiga k escoge ir al siguiente nodo con una probabilidad que se calcula como:

$$p_{ij}^k = \frac{(\tau_{ij})^\alpha * (\eta_{ij})^\beta}{\sum_{l \in N_i^k} (\tau_{il})^\alpha * (\eta_{il})^\beta} \qquad si \quad j \in N_i^k \tag{1.3}$$

donde N_i^k es la vecindad alcanzable por la hormiga k cuando se encuentra en el nodo i, los parámetros alfa (α) y beta (β) controlan el proceso de búsqueda. Para α=0 se tiene una búsqueda heurística estocástica clásica, mientras que para $\beta = 0$ solo el valor de la feromona tiene efecto. Un valor de $\alpha < 1$ lleva a una rápida situación de convergencia (stagnation). P_{ij}^k es un vector de probabilidades de visitar cada uno de los nodos de la vecindad (N_i^k) por la hormiga *k*, τ_{ij} el valor del elemento i, j en la matriz de feromona y η_{ij} , se denomina función de visibilidad o función heurística, la cual depende totalmente de las características del problema que se va a resolver.

Sistema de Colonias de Hormigas

El Sistema de Colonias de Hormigas (*Ant Colony System, ACS*) es uno de los primeros sucesores del AS, el cual presenta tres modificaciones importantes con respecto a su predecesor:

1. El ACS usa una regla de transición distinta y más agresiva, denominada regla proporcional pseudo-aleatoria. Sea k una hormiga situada en el nodo r, $q_0 \in [0,1]$ un parámetro y q un valor aleatorio en [0,1], el siguiente nodo i se elige aleatoriamente mediante la siguiente distribución de probabilidad:

   si $q \leq q_0$:

$$P_{ij}^k = \begin{cases} 1, & si \quad j = \max_{j \in N_i^k} \left\{ \tau_{ij} * \eta_{ij}^{\beta} \right\} \\ 0, & otro\ caso \end{cases} \tag{1.4}$$

   sino ($q > q_0$):

$$P_{ij}^k = \begin{cases} \dfrac{(\tau_{ij})^{\alpha} * (\eta_{ij})^{\beta}}{\sum_{l \in N_i^k} (\tau_{il})^{\alpha} * (\eta_{il})^{\beta}} & si \quad j \in N_i^k \\ 0, & otro\ caso \end{cases} \tag{1.5}$$

   Como puede observarse, la regla tiene una doble intención: cuando $q \leq q_0$, la hormiga explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y los rastros de feromona. Sin embargo, si $q > q_0$ la hormiga explora el espacio de búsqueda seleccionando un nodo aleatorio del conjunto de sus posibles movimientos.

2. Las hormigas aplican una actualización en *línea paso a paso* de los rastros de feromona que favorece la generación de soluciones distintas a las encontradas.

   Cada vez que una hormiga viaja por una arista a_{ij}, aplica la regla:

$$\tau_{ij} \leftarrow (1-\varphi) * \tau_{ij} + \tau_0 \tag{1.6}$$

   donde $\varphi \in (0,1]$ es un segundo parámetro de decremento de feromona. Como puede verse, la regla de actualización en *línea paso a paso* incluye tanto la evaporación de feromona como la deposición de la misma. Debido a que la cantidad de feromona depositada es muy pequeña, la aplicación de esta regla hace que los rastros de feromona entre las conexiones recorridas por las hormigas disminuyan. Así, esto lleva a una técnica de exploración adicional del ACS ya que las conexiones atravesadas por un gran número de hormigas son cada vez menos atractivas para el

resto que se encuentra trabajando en la iteración actual, lo que ayuda claramente a que no todas las hormigas sigan el mismo camino.

3. Se realiza una actualización de feromona fuera de línea de los rastros. Para llevarla a cabo, el ACS sólo considera una hormiga concreta, la que generó la mejor solución global, $S_{mejor\text{-}global}$ (aunque en algunos trabajos iniciales se consideraba también una actualización basada en la mejor hormiga de la iteración, en ACO casi siempre se aplica la actualización por medio de la mejor global).

   La actualización de la feromona se hace evaporando primero los rastros de la misma en todas las conexiones utilizadas por la mejor hormiga global (es importante recalcar que, en el ACS, la evaporación de feromona sólo se aplica a las conexiones de la solución, que es también la usada para depositar feromona) tal como sigue:

$$\tau_{ij} \leftarrow (1-p) * \tau_{ij} \quad \forall a_{ij} \in S_{mejor-global} \tag{1.7}$$

   A continuación se deposita feromona a los arcos que pertenecen a la mejor solución encontrada hasta el momento usando la regla:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta t \quad \forall a_{ij} \in S_{mejor-global} \tag{1.8}$$

   Donde $\Delta t = f(C(S_{mejor-global}))$, es decir la cantidad de feromona está en dependencia de la calidad de la mejor solución encontrada hasta el momento $C(S_{mejor-global})$.

1.3 Conclusiones parciales

A partir de la consulta bibliográfica realizada para presentar este capítulo se pueden extraer las conclusiones siguientes:

- El análisis del estado del arte ha permitido conocer la situación actual en que se encuentra el estudio sobre los modelos de razonamiento funcional existentes. El análisis bibliográfico evidencia que existen diversos estudios acerca de teorías y modelos de razonamiento funcional para problemas de diseño, estos modelos han ido evolucionando con los años y proponiendo nuevas formas de encontrar soluciones, pero no se encuentran algoritmos que permitan solucionar problemas de configuración basados en alguno de ellos.
- El modelo propuesto por Chakrabarti y Bligh en 2001 se basa en la idea de ir encontrando soluciones parciales, es decir, el problema no necesita ser resuelto como un todo, sino que puede ser dividido en partes. Esta idea de ir incorporando en cada paso nuevas estructuras que aportan a lo que será la solución final fue la seleccionada para aplicar en la búsqueda de soluciones.

- Debido a la complejidad computacional del proceso de construcción de configuraciones cuando la cantidad de componentes crece y al objetivo que se persigue, que es minimizar la cantidad de componentes presentes en la solución, se hace necesario el empleo de métodos heurísticos.
- Se analizó la aplicabilidad del algoritmo Ascensión de Colinas, por ser un algoritmo sencillo y de bajo costo computacional, además se analizaron la optimización basada en colonias de hormigas y los algoritmos genéticos que caen en la categoría de algoritmos bioinpirados, los cuales han cobrado auge en la comunidad científica y están siendo aplicados en una amplia gama de campos.

Capítulo 2 Algoritmos heurísticos para resolver problemas de configuración

A partir de la revisión bibliográfica realizada acerca de los problemas de razonamiento funcional, los modelos existentes para resolver problemas de configuración y los métodos heurísticos, en este capítulo se proponen tres algoritmos, cada uno basado en un método heurístico diferente, para dar solución a este tipo de problemas usando bases de datos de componentes.

Las bases de datos nos brindan, para cada problema, el consumo máximo del que se dispone, las funciones que se deben obtener, las cuales usualmente se llaman necesidad y aquellos componentes disponibles para configurar, cada uno con las funciones que ofrece y aquellas que necesita para poder ofrecer estas.

Se pretende, en todos los casos, minimizar la cantidad de componentes que logre satisfacer los requerimientos sin sobrepasar el consumo ofrecido.

2.1 Ascensión de colinas aplicado a problemas de configuración

Para aplicar el algoritmo de ascensión de colinas en la solución de problemas de configuración, teniendo en cuenta que la idea es partir de una base de datos de componentes, contando además un consumo máximo disponible y ciertas funciones que satisfacer, se hace uso de una representación binaria (ceros y unos), es decir, las cadenas, que representarán posibles configuraciones y tendrán tamaño igual a la cantidad de componentes presentes en la base de datos, contarán en cada posición con un valor (0 o 1) que indica si el componente está presente o no en la solución.

Por ejemplo, la cadena 0110 no es más que una configuración para una base de datos de 4 componentes (A, B, C y D), donde solo están presentes los componentes B y C.

El algoritmo ascensión de colinas comienza su trabajo generando una solución y evaluándola para a partir de ella comenzar a buscar una mejor que la reemplace. En este caso, se genera una primera configuración aleatoria, que posteriormente se evalúa para saber cuán buena es con respecto a la necesidad que se desea satisfacer. Para esto primero se debe obtener de la base de datos del problema que se está resolviendo cuál es el objetivo, que tiene la siguiente forma:

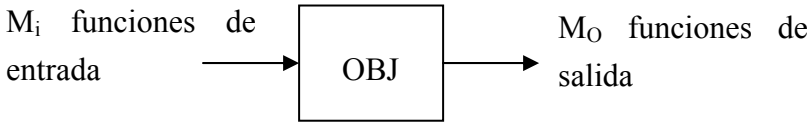


Figura 2.1. Representación del objetivo

Aquí, las funciones de salida constituyen la necesidad y las funciones de entrada el consumo máximo disponible. Cada configuración generada agrupa en sus funciones de salida todas aquellas ofrecidas por el conjunto de componentes que la constituyen, y lo mismo se hace con las funciones de entrada, que son todas aquellas que necesitan ser consumidas por el grupo de componentes presentes en la configuración.

En esta estructura resultante existen funciones de salida de algunas componentes necesarias como entradas de otras componentes incluidas en la misma; de modo que se establece un balance interno por el cual se eliminarán salidas de la estructura al ambiente externo así como necesidades. La estructura balanceada está entonces lista para ser evaluada, para lo cual se tienen en cuenta dos puntos importantes:

- Por cada función de salida de la configuración que coincida con M_o sumar 1.
- Por cada función de entrada de la configuración no satisfecha por las entradas M_i restar 1.

Por ejemplo, si se tiene la base de datos siguiente:

Tabla 2.1. Base de datos de componentes

Componente	**Consume**	**Ofrece**
A	f1, f3	f2, f5
B	f4	f3, f8
C	f1, f5	f6
D	f3	f2
Necesidad	f6, f8	
Consumo Máximo	f1, f1, f4	

El objetivo es: Posible configuración:

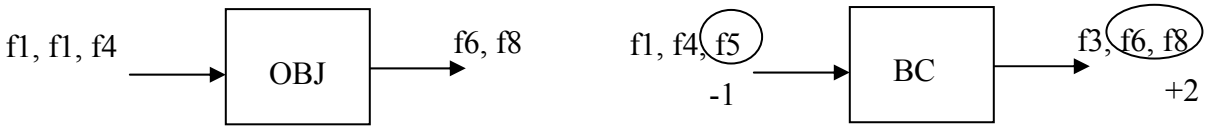


Figura 2.2. Ejemplo de objetivo y una posible configuración

Si se calcula el valor de la configuración BC el resultado es 1, pues al configurar los componentes B y C, la estructura resultante es capaz de ofrecer las funciones presentes en la necesidad (f6 y f8). Pero esta combinación necesita consumir la función f5, la cual no aparece en el consumo disponible y por tanto es una función que aún queda sin satisfacer. Este valor de la configuración también se denomina aptitud.

Una vez generada y evaluada la primera configuración, se comienza el proceso de ir variando componentes para obtener nuevas configuraciones a partir de ella. Variar componentes no es más que tomar dicha configuración inicial y cambiar el valor de una de sus posiciones. En la representación que estamos utilizando, esto implica que un 0 se cambiará por un 1 o viceversa, que en otros términos quiere decir que un componente que estaba presente en la solución inicial dejará de estar en la nueva o que aparecerá en ella sin haber estado en la original.

Si al variar un componente y evaluar la nueva configuración, ésta tiene mejor aptitud, se pasa la misma a nodo actual y se continúa el proceso, es decir, se hace una escalada hacia la primera opción que mejore el estado actual.

El estado final estará determinado por la obtención de una configuración que satisfaga la necesidad y que cuente con la menor cantidad de componentes posible. El algoritmo termina cuando no hay mejorías, es decir, cuando el mejor resultado alcanzado no ha sido mejorado durante una cierta cantidad de iteraciones. Por ejemplo, si se tiene la base de datos de componentes que se muestra en la tabla 2.1 el algoritmo funciona de la siguiente manera:

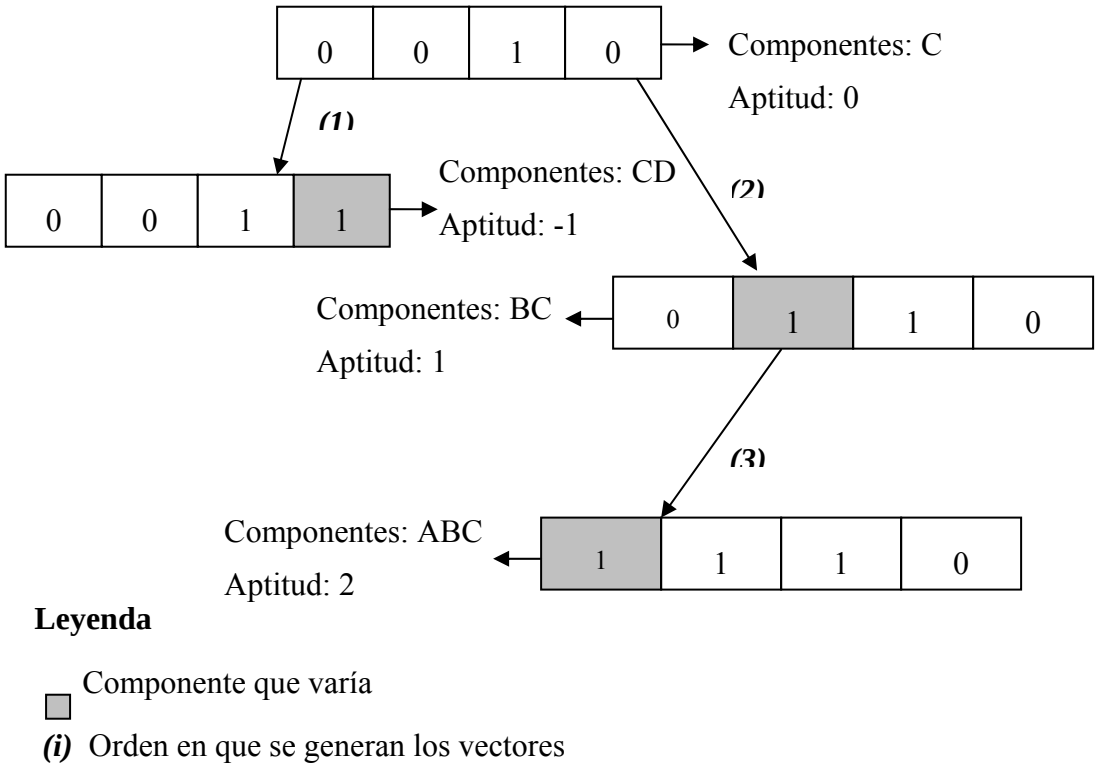


Figura 2.3. Ejemplo del funcionamiento del algoritmo Ascensión de Colinas

Encontrándose como solución la configuración de los componentes A, B y C. Si se compara esta posible solución con la estructura BC que se analizó anteriormente, se puede comprobar que en este caso, agregando el componente A, se logra que la función f5 que anteriormente quedaba pendiente (Fig. 2.3), quede ahora satisfecha (Fig 2.4).

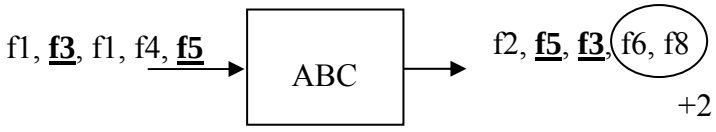


Figura 2.4. Ejemplo de una configuración

En este caso aparecen subrayadas y en negrita las funciones que se anulan, pues son satisfechas al combinar estos componentes, y circuladas las que posibilitan que esta combinación sea una solución al problema inicial.

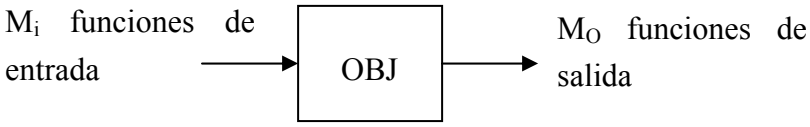
El pseudo código del algoritmo implementado es el siguiente:

```
Fase de inicialización
 Inicializar cantidad de iteraciones NI
 Inicializar el objetivo
```

```
Ca ← Generar configuración actual
Ea ← Evaluar Ca
Mientras NI ≠ 0
     Cn ← Generar nueva configuración a partir de Ca
     En ← Evaluar Cn
     Si En > Ea entonces
          Ca ← Cn
          Ea ← En
     NI ← NI – 1
Imprimir mejor solución (Ca)
```

2.2 Algoritmo Genético aplicado a problemas de configuración

La aplicación de un Algoritmo Genético en la solución de problemas de configuración partiendo de un fichero de componentes requiere como primer paso determinar el objetivo o configuración deseada, ésta tiene la misma forma que el objetivo en el algoritmo Ascensión de Colinas:



Donde las funciones de entrada representan el consumo máximo del que se dispone y las funciones de salida aquellas características que se desean obtener.

Diseño de los cromosomas

Una vez determinado el objetivo se pasa a la creación de los cromosomas, para esto es necesario realizar su diseño. En este caso se utiliza la representación tradicional del cromosoma como una cadena binaria (de 0 y 1). La cadena tendrá una longitud igual a la cantidad de componentes que tiene la base de datos utilizada. Cada posición en el cromosoma será un gen cuyo valor (0 ó 1) indicará si está presente o no en la configuración representada por el cromosoma.

Población inicial

Para comenzar el algoritmo es necesario crear una población inicial, para lo cual existen fundamentalmente dos opciones:

a) Generación aleatoria de individuos

b) Sembrado de individuos

En este caso se parte de una población inicial creada de forma aleatoria, donde cada cromosoma tiene una aptitud en base a lo que es capaz de ofrecer y lo que necesita con respecto al objetivo.

Función de aptitud

Cada cromosoma tiene un valor, denominado aptitud, que se calcula en base a lo que es capaz de ofrecer y lo que necesita con respecto al objetivo. Para el cálculo de este valor de aptitud para un cromosoma se tiene en cuenta cuales componentes están incluidas, cuales funciones ofrece y necesita y cual es el balance interno resultante (tal y como se explicó anteriormente).

El cálculo de la aptitud para un cromosoma X con respecto al objetivo OBJ se realiza de la forma siguiente:

- Por cada función de salida de X que coincida con M_o sumar 1.
- Por cada función de entrada de X no satisfecha por las entradas M_i restar 1.

Al calcular la aptitud del cromosoma C_1 con respecto al objetivo OBJ, el resultado sería 1, pues al estar presentes en C_1 tanto f4 como f7, ambas implican un aumento en 1, pero al no estar presente en el objetivo f5, y si estarlo en C_1, es una función que queda sin satisfacer, produciendo una disminución en 1 en el cálculo de la aptitud del cromosoma.

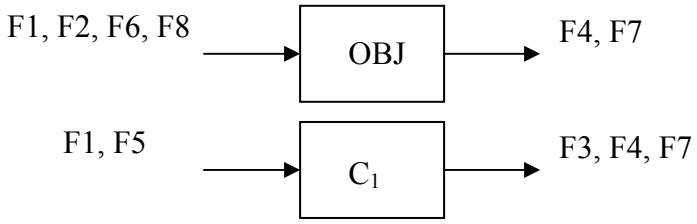


Fig. 2.5. Ejemplo del cálculo de aptitud para un cromosoma

Aplicación de los operadores genéticos

Operador de Selección

Después de calculadas las aptitudes para todos los cromosomas que conforman la población es aplicado el operador de selección. En este caso se utiliza el método de la ruleta, que funciona de la manera siguiente:

- Se evalúa la función en cada individuo, obteniendo para cada uno su aptitud.
- Se calcula la suma de todas estas evaluaciones.

- Se calcula la probabilidad de cada uno para ser seleccionado como la fracción de esa suma total que le corresponde.
- Se generan números aleatorios y se seleccionan los individuos.

Ejemplo:

Suponga que se desea maximizar la función x^2 en el intervalo [0…31]. Para lograr esto es necesario codificar los individuos de la población, en este caso los puntos. Como son 32 puntos distintos y se codificará en una cadena de valores binarios solo se necesita una cadena de longitud 5.

Para cuatro (4) individuos se obtiene entonces la tabla siguiente:

Individuo	**Cadena**	**Valor x**	**f(x)=x^2**	**f/SUMA**
1	01101	13	169	0.14
2	11000	24	576	0.49
3	01000	8	64	0.06
4	10011	19	361	0.31
SUMA			1170	1.0
PROM (Promedio)			293	0.25
MAX			576	0.49

La selección se realiza como sigue:

Se utiliza el valor de f(x)/SUMA como probabilidad y se forman intervalos donde cada individuo tendrá un tramo según su valor de probabilidad:

1- desde 0 hasta 0.14

2- desde 0.14 hasta 0.63=0.14+0.49

3- desde 0.65 hasta 0.69=0.63+0.06

4- desde 0.69 hasta 1.00=0.69+0.31

Cuando se genera un número aleatorio, la probabilidad de que esté en el intervalo de un individuo estará en función del tamaño del mismo y por lo tanto de su adaptación al medio.

Como se observa no se realiza un proceso determinístico en el momento de la selección lo cual permite una simulación más exacta del proceso natural. El proceso de reproducción no solo garantiza que el individuo mejor adaptado tenga una participación más fuerte en la formación de la nueva generación sino que esta magnitud también está fijada por la proporción de la adaptación de ambos.

Operador de Cruzamiento

Después de aplicada la selección y obtenidos los mejores individuos pasamos se realiza el cruzamiento. Para esto se seleccionó el cruce de un punto, el cual elige aleatoriamente un punto de ruptura en los padres para luego intercambiar sus bits.

Por ejemplo, si se tienen los cromosomas:

a) 101110 Madre

b) 111001 Padre

Seleccionando este punto de cruce, o sea, 4, el resultado es:

Hijo 1: 101001

Hijo 2: 111110 Los genes que aparecen subrayados son los pertenecientes a la madre

Se puede apreciar que el cromosoma denominado Madre es una configuración compuesta por los componentes ACDE, mientras que en el Padre están presentes los componentes ABCF. Después de realizado el cruzamiento, aparecerían en el hijo 1 los componentes ACF y en el 2 se encuentran todos excepto el F.

O sea, que para obtener la descendencia en cromosomas de longitud N después de tener el punto de cruce X, el hijo 1 toma la información desde 1 hasta X-1 de la Madre y desde X hasta N del padre, el hijo 2 por su parte, toma desde 1 hasta X-1 del padre y desde X hasta N de la madre.

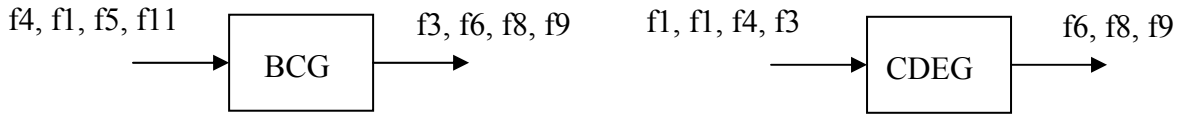
Como se puede observar, si el número generado es 1, la primera porción de la cadena quedará vacía (desde el 1 hasta el anterior a X=1, o sea desde 1 hasta 0) lo cual indica que en realidad no se está realizando un intercambio de información, pues cada hijo será exactamente igual a uno de sus padres. Esto no puede considerarse un error pero sí es importante tenerlo en consideración en el funcionamiento del AG.

Este cruzamiento no tiene necesariamente que provocar la obtención de un organismo mejor. Su misión principal es la exploración de un nuevo camino, o sea, crear a partir de la combinación de lo ya conocido. El siguiente ejemplo nos muestra la realización del cruzamiento para una base de datos de componentes.

Tabla 2.2: Ejemplo de una base de datos de 7 componentes

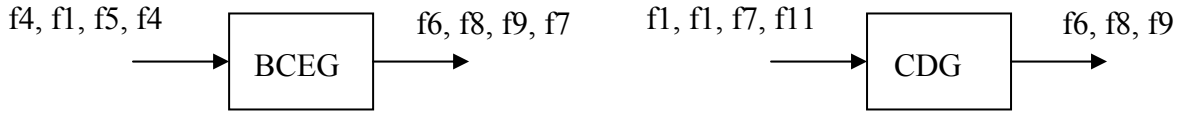
Componente	Consume	Ofrece
A	f1, f3	f2, f5
B	f4	f3
C	f1, f5	f6, f8
D	f1, f7	f5
E	f4, f9	f7, f11
F	f4	f7, f10
G	f11	f9
Necesidad	f6, f8	
Consumo Máximo	f1, f1, f4	

Si como madre se tiene el cromosoma 0110001 (BCG) y como padre el 00111 (CDEG), los cuales se muestran a continuación con sus respectivas aptitudes:



Aptitud: 1 Aptitud: 2

Y seleccionáramos como punto de cruce la posición 5, se obtienen como resultado los hijos 0110101 (BCEG) y 0011001 (CDG). Una vez calculadas sus aptitudes, podemos apreciar que no se obtienen precisamente mejores individuos de este proceso.



Aptitud: 1 Aptitud: 1

Es importante recordar que se trabaja con una probabilidad de cruzamiento, es decir, que para que este proceso se realice es necesaria la generación de dos números aleatorios, el primero para decidir si el proceso se realiza o no en base a la probabilidad utilizada (el número generado tiene que ser menor que la probabilidad), y el segundo en caso de que el resultado anterior sea positivo, para obtener el punto por donde será realizado el cruce. Este punto no es fijo para todos los elementos de una generación.

Operador de Mutación

Una vez terminado el cruzamiento se pasa al proceso de mutación, el cual es similar al anterior, pues la decisión de si un cromosoma mutará o no se toma en base a la probabilidad de mutación con la que se trabaja. En caso positivo, se escoge de forma aleatoria el gen a mutar y su valor es intercambiado. Si era 0 se cambia a 1 o viceversa, lo que indica que un componente deja de estar presente en una configuración o que es incorporado en la misma. Este proceso tampoco garantiza que los individuos obtenidos sean siempre mejores.

Después de aplicados estos operadores, se evalúa la nueva población obtenida para luego realizar la comparación con la inicial con vistas a determinar cual será la que permanecerá en el paso a la próxima generación. En este caso el criterio para tomar esta decisión es escoger la que mejor promedio de aptitud tenga entre todos sus individuos. Este proceso se repite hasta que se cumpla el criterio de parada, que en este caso es la cantidad de generaciones.

Es importante señalar que puede ocurrir, debido a las características del problema, que en la población aparezcan cromosomas que no constituyen una solución al problema original porque no llegan a satisfacer todos los requerimientos. Cuando estos casos se presentan los individuos son penalizados, asignándoseles valores de aptitud muy bajo.

El algoritmo en pseudo código es el siguiente:

```
Fase de inicialización
 Inicializar cantidad de generaciones NG
 Inicializar las probabilidades de cruzamiento y mutación
 Inicializar el tamaño de la población a utilizar
 Inicializar el objetivo
 Crear la población inicial aleatoria
 Calcular aptitud de los cromosomas de la población inicial

Repetir NG veces
```

```
  Aplicar Selección a la población
  Aplicar Cruzamiento a la población
  Aplicar Mutación a la población
  Calcular promedio de aptitud de la población resultante
  Comparar población resultante con la inicial y seleccionar la mejor
  para repetir el procedimiento
```

Imprimir mejor solución (cromosoma de mejor aptitud)

2.3 Sistemas de Colonias de Hormigas aplicado a problemas de configuración

Para aplicar la optimización basada en colonias de hormigas en la implementación del razonamiento funcional para resolver problemas de configuración se selecciono el algoritmo Sistemas de Colonias de Hormigas (*Ant Colony System, ACS*) por ser uno de los que mejor desempeño ha mostrado en esta familia. Se construye inicialmente un grafo no dirigido G=(N, A), donde el conjunto de nodos es igual a la cantidad de componentes que presenta la base de datos del problema a resolver, representando los arcos la conexión funcional entre cada uno de los componentes. En este tipo de problema es importante el orden en que son escogidos los mismos, pues a medida que van siendo elegidos se van actualizando las funciones a buscar, por lo que en este caso se tiene una matriz de feromona, que es lo mismo que asociar la feromona a los arcos del grafo. Esta matriz se inicializa mediante alguno de los criterios presentados en el capítulo anterior.

Otro elemento a definir cuando se aplica ACO en la solución de problemas es el relacionado con los nodos iniciales en cada iteración del algoritmo. En este caso se seleccionan como nodos iniciales aquellos que ofrecen la necesidad inicial del problema o alguna función que aparezca en ella, y se ubican las hormigas de forma aleatoria en alguno de ellos, para luego comenzar el proceso de búsqueda hacia atrás como fue explicado anteriormente.

Después que todas las hormigas son ubicadas en su componente o nodo inicial, comienzan a moverse por el grafo según las expresiones (1.4) y (1.5). Para este problema la función heurística es:

$$\eta_{ij} = \frac{1}{100 - F(X_n, M_{oj})} \qquad (2.1)$$

donde F es una función en $X_{n,}$ que representa las funciones necesitadas actualmente y M_{oj} las funciones que el componente j ofrece. F retorna la cantidad total de funciones que j es capaz de

ofrecer de la necesidad actual y 100 se asume que es la cantidad máxima de funciones (o combinaciones de ellas) que se puede ofrecer.

En este problema la vecindad N_i del nodo *i,* está compuesta por aquellos componentes que son capaces de ofrecer la necesidad actual que presenta la hormiga que está realizando el movimiento, o parte de ella. Es decir, que en caso de que una hormiga decida moverse hacia el mejor nodo (el número aleatorio generado fue menor que el q_0 utilizado), entonces se escoge dentro de la vecindad aquel componente que ofrezca la mayor cantidad de funciones de las que se encuentran en la necesidad actual. En caso contrario, se escoge un nodo cualquiera dentro de los que aparecen en su vecindad.

Una vez escogido el próximo componente, este se introduce en la lista tabú de la hormiga en cuestión. Luego se procede a incrementar los valores de feromona con una actualización *paso a paso* según la expresión 1.6. Este proceso es repetido por todas las hormigas de la colonia, y las mismas terminan su recorrido cuando la necesidad sea satisfecha o lo que es lo mismo, el conjunto de las funciones a buscar se haga vacío. Por ejemplo, si se tienen los siguientes componentes (que constituyen parte de una base de datos):

Tabla 2.3. Parte de una base de datos de componentes

Componente	Consume	Ofrece
A	F1, F13	F5, F3
B	F4, F12	F7, F10
C	F1, F5, F3	F6
D	F1, F7	F5
…		

Si una hormiga *i* tuviera como nodo inicial el nodo C y la necesidad actual contara con las funciones F8, F5, F3, entonces su vecindad sería: N_i = (A, D); en el caso de A porque ofrece las funciones F5 y F3 y D por ofrecer F5.

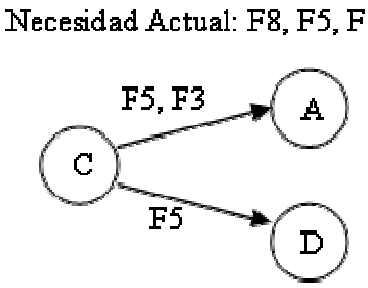


Figura 2.6. Ejemplo de vecindad para una hormiga

Si en esta situación la hormiga (de acuerdo a las reglas de movimiento del algoritmo) fuera a realizar un movimiento hacia el mejor nodo, el escogido sería A, pues es capaz de satisfacer una mayor cantidad de funciones de la necesidad actual.

Cuando todas las hormigas han terminado su recorrido, las soluciones encontradas son evaluadas para tener almacenada la mejor reportada hasta el momento (*mejor solución global*), que es aquella que minimiza el objetivo. En este caso se selecciona la que menor cantidad de componentes incorpore; pero, por ejemplo, se pudiera utilizar el costo total de la configuración.

Por ejemplo, si se tiene la base de datos que aparece en la Tabla 2.2 y se aplicara el algoritmo, la única opción a escoger como nodo inicial es el nodo C, pues únicamente el ofrece las funciones presentes en la necesidad inicial. Una vez terminada la ejecución del método, las hormigas pueden haber encontrado algunas de las soluciones siguientes:

Hormiga 1: CAB

Hormiga 2: CDEG

Hormiga 3: CDF

En este caso aparecen dos soluciones de tamaño 3 y una de tamaño 4, siguiendo el criterio de minimizar la cantidad de componentes en la solución, tanto la encontrada por la hormiga 1 como la encontrada por la 3 puede ser escogida como la mejor solución global.

Luego se procede a realizar la evaporación de los rastros de feromona pertenecientes a esa mejor solución encontrada hasta el momento según expresión 1.7. Una vez evaporado el camino, se realiza también una actualización de los rastros de feromona asociados a esta solución, proceso que es llamado *actualización global de los rastros de feromona* y se realiza según expresión 1.8.

El algoritmo trabaja con una serie de parámetros cuyos valores son determinados por el usuario, o sea, parámetros de entrada que pueden variar de una ejecución a otra del algoritmo, estos son:

Feromona Inicial, Constante de Evaporación, cantidad de ciclos, cantidad de hormigas a utilizar, Beta, y q_0.

El pseudo-código del algoritmo es el siguiente:

Fase de inicialización

Inicializar contador de ciclos NC

Inicializar total de hormigas

Inicializar solución global L_{Global}=ø

Inicializar Matriz de Feromonas:

Para cada arco (i, j)→ Valor inicial de $\tau_{ij} = \tau_0$

Repetir NC veces

Para cada hormiga

Elegir nodo inicial

Repetir hasta que cada hormiga termine su recorrido

Para cada hormiga:

Elegir próximo nodo según ecuaciones 1.4 y 1.5

Insertar nodo escogido en la solución parcial

Actualizar las funciones a buscar

Actualizar feromona paso a paso según ecuación 1.6

Buscar la mejor solución del ciclo L_k

Comparar con la mejor solución existente

Si ($L_k < L_{global}$) Actualizar L_{global} con L_k

Actualizar feromona global según ecuaciones 1.7 y 1.8

Imprimir mejor solución L_{Global}

2.4 Conclusiones parciales

Los nuevos algoritmos introducidos en este capítulo proponen una nueva forma de enfrentar los problemas de configuración; los mismos se basan en la aplicación de métodos heurísticos, pues debido a las características del problema, donde se persigue minimizar la cantidad de componentes presentes en la solución, este puede clasificarse como problema de optimización, sin que se haya

encontrado en la revisión bibliográfica realizada, la utilización de este tipo de métodos para dar solución al problema.

Los métodos heurísticos utilizados fueron Ascensión de Colinas, Algoritmo Genético y Sistema basado en colonias de hormigas, realizándose en este capítulo una detallada descripción de como fue modelado cada uno de ellos para ser aplicado al problema en cuestión.

Capítulo 3 Análisis de los resultados obtenidos

Este capítulo está dedicado a reflejar los resultados de la aplicación de los algoritmos propuestos a distintas bases de datos de componentes. Se explicará inicialmente el proceso de construcción de las bases de datos a utilizar y luego se mostrará el desempeño de los métodos haciendo uso de las mismas.

Los algoritmos que aquí proponemos se basan todos en el modelo propuesto por Chakrabarti y Bligh en el año 2001, es decir, que todos nos permiten, contando con un consumo máximo y una necesidad, partir de una base de datos de componentes para encontrar la menor cantidad de ellos que logre satisfacer los requerimientos sin sobrepasar el consumo disponible.

Se analizan los valores de los parámetros a utilizar en cada método heurístico, realizando pruebas con distintas variantes en aras de encontrar los más adecuados en la obtención de buenas soluciones.

3.1 Definición de las bases de datos de componentes

Al no contar para este problema con bases de datos internacionales que se pudieran utilizar para validar el desempeño de los algoritmos que se proponen, fue necesario crear bases de datos de prueba que permitieran aplicar los algoritmos para analizar los resultados obtenidos.

Por tanto, estudiando las características del problema, se decidió implementar un algoritmo que permitiera, conociendo la cantidad de funciones y la cantidad de componentes con las que se desea trabajar, generar aleatoriamente una base de datos, donde a cada componente le son asignadas las funciones que ofrece y las que consume.

En el caso de la necesidad y el consumo máximo, que también son generados de forma aleatoria, se maneja en la implementación que las funciones que aparezcan en uno no lo hagan en el otro, es decir, que no se generen funciones que se anulen

Pseudo-código del algoritmo:

```
Dadas la cantidad de funciones (CF) y de componentes a utilizar (CC)
Generar la necesidad inicial
Generar el consumo máximo disponible
Repetir CC veces
 Generar Oferta del componente actual
 Generar Consumo del componente actual
Imprimir en fichero los datos generados
```

3.2 Resultados Experimentales

Todos los resultados experimentales obtenidos en este epígrafe fueron desarrollados en una P4 Intel a 2.8 GHz de velocidad, 1GB de memoria RAM, con sistema operativo Windows 2003. La implementación de los algoritmos se realizó en el lenguaje de programación Java, para lo cual se utilizó el Eclipse como entorno de desarrollo, en todos los casos se realizaron 10 corridas de cada algoritmo.

Las tablas que muestran los resultados cuentan con dos columnas fundamentales para cada variante ejecutada, la primera llamada "Número de componentes", que como su nombre lo indica, nos muestra la cantidad de componentes que aparecen en la solución obtenida y la segunda columna es "Tiempo", que muestra (en milisegundos) el tiempo que el algoritmo necesitó para obtener esa solución.

3.2.1 Resultados obtenidos por Ascensión de Colinas

Para la ejecución del algoritmo Ascensión de Colinas se tuvieron en cuenta dos variantes, en las cuales el parámetro que varía es la cantidad de iteraciones a ejecutar. En la primera variante se trabajó con una cantidad de iteraciones de 500 y en la segunda este valor se aumentó a 1000.

Tabla 3.1. Resultados obtenidos por el algoritmo Ascensión de Colinas

	Variante 1		**Variante 2**	
	Número de Componentes	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**
Comp30.fr	12	9797	11	18656
Comp50.fr	24	13547	20	29172
Comp70.fr	27	72640	23	16754
Comp100.fr	36	75219	32	185141
Comp120.fr	42	168469	48	178938
Comp150.fr	59	128765	54	271438
Comp170.fr	67	186453	63	386719

Como se puede observar en los resultados que muestra la tabla y en el gráfico mostrado en la Figura 3.1, las mejores soluciones aparecen cuando la cantidad de iteraciones ejecutadas es mayor. Cuando el tamaño de las bases de datos aumenta, es decir, cuando la cantidad de componentes a configurar es mayor, pues resulta más difícil que el algoritmo logre obtener resultados con el número de iteraciones utilizado, como es el caso de las bases de datos de tamaño 200 en adelante (que no aparecen en la tabla), es por esto que después de analizar el comportamiento del algoritmo Ascensión de Colinas, que es un método sencillo, de bajo costo computacional y que nos sirvió de prueba inicial, se decidió aplicar otros métodos heurísticos que fueran capaces de obtener mejores soluciones.

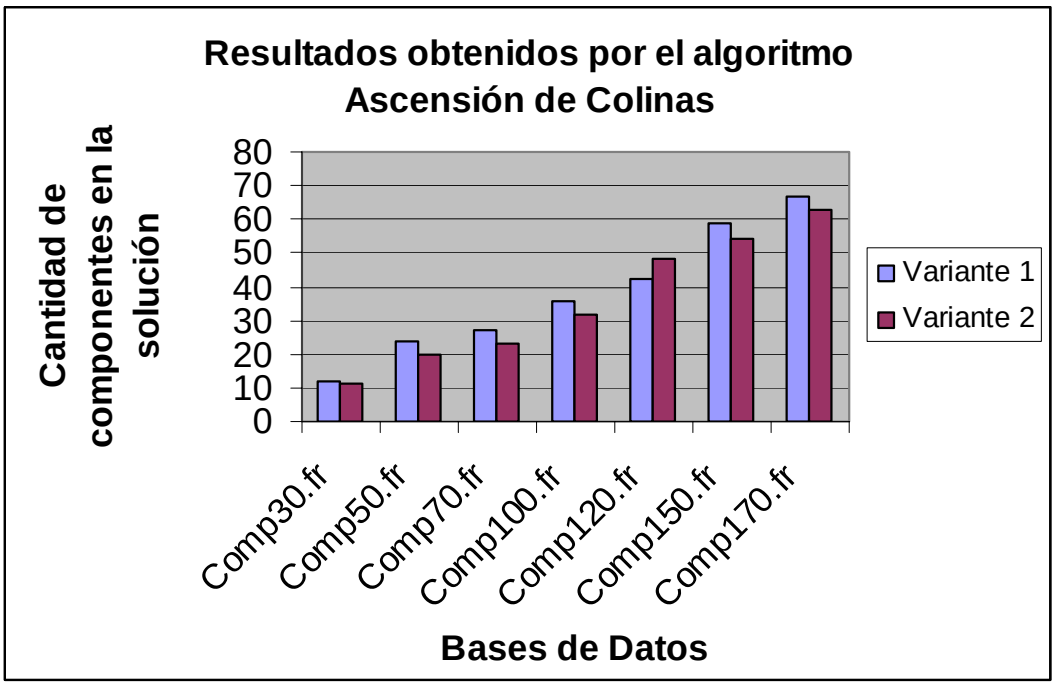


Figura 1. Resultados obtenidos por el algoritmo Ascensión de Colinas

3.2.2 Resultados obtenidos por el Algoritmo Genético

Para la ejecución del AG se requiere de varios parámetros, cuya determinación no es una tarea trivial, en tanto que no puede afirmarse que tales o cuales valores de ellos sean los más apropiados. Si bien no existen valores que sean los mejores en todos los casos, si hay algunas recomendaciones en torno a ello, como en [48].

Probabilidad de cruzamiento

Se tuvo en cuenta para la implementación la recomendación de que la probabilidad de ocurrencia del

cruzamiento debe ser alta para lograr una elevada introducción de nuevas soluciones del problema en cuestión.

La combinación de individuos que implica la aplicación de este operador genético provoca la aparición de nuevos elementos, los cuales no siempre mejoran a sus antecesores, pero obviamente logran diversidad en la población.

Probabilidad de mutación

Algunas de las razones que pueden motivar a incorporar mutaciones son:

- Desbloqueo del algoritmo: Si el algoritmo se bloqueó en un mínimo parcial, una mutación puede sacarlo al incorporar nuevos fenotipos de otras zonas del espacio
- Incrementar el número de saltos evolutivos
- Enriquecer la diversidad genética

Para el algoritmo es importante tener control de la mutación ya que estos cambios pueden tener tanto un efecto conveniente como perjudicial, el cambio de información puede provocar una mejoría importante en un individuo de la población proporcionándole una característica esencial o puede hacer que pierda una que ya tenía.

Tamaño de la población

Muchos trabajos se han escrito relativos a la influencia del tamaño de la población en la convergencia del AG. En principio, es lógico pensar que el trabajo con poblaciones pequeñas corre el riesgo de representar pobremente el espacio de soluciones. Por otro lado, las poblaciones de gran tamaño consumen mayor tiempo computacional. Sobre esta disyuntiva y como un trabajo teórico, Goldberg obtuvo en su investigación que el tamaño óptimo de una población de cadenas binarias, crece exponencialmente con la longitud de la cadena [25]. Sin embargo, en diferentes resultados empíricos muchos autores sugieren tamaños de poblaciones tan pequeños como 30 individuos.

Algunos aspectos que se pueden extraer de las investigaciones analíticas y empíricas son los siguientes:

1. Incrementando la probabilidad de cruzamiento se incrementa la recombinación de bloques de construcción, pero también se puede incrementar la destrucción de buenas cadenas.
2. Incrementando la probabilidad de mutación se tiende a transformar la búsqueda genética en una búsqueda aleatoria, pero esto también permite reintroducir pérdida de material genético.

3. Incrementando el tamaño de la población se incrementa la diversidad y reduce la probabilidad de que el AG haga una convergencia prematura, pero esto también incrementa el tiempo requerido para que la población converja a las regiones optimales en el espacio de búsqueda.

Para seleccionar los parámetros se tuvieron en cuenta también los estudios de De Jong (1975) donde se sugiere:

- Tamaño de la Generación: debe ser un valor moderado de entre 20 y 100 individuos.
- Probabilidad de Cruzamiento: alta.
- Probabilidad de Mutación: baja, inversamente proporcional al tamaño de la generación.

Después de analizar los estudios mencionados acerca de la selección de los parámetros adecuados para la ejecución del algoritmo se determinó analizar diferentes variantes, utilizando en todos los casos como condición de parada procesar un número determinado de generaciones. Las variantes estarían dadas por los valores asignados a los distintos parámetros, variando fundamentalmente:

- Tamaño de población
- Cantidad de generaciones

Variante 1: Parámetros utilizados

- Tamaño de población = 30
- Probabilidad de cruzamiento = 0.85
- Probabilidad de mutación = 0.01
- Número de generaciones a ejecutarse = 100

Variante 2: Parámetros utilizados

- Tamaño de población = 30
- Probabilidad de cruzamiento = 0.85
- Probabilidad de mutación = 0.01
- Número de generaciones a ejecutarse = 300

Variante 3: Parámetros utilizados

- Tamaño de población = 80
- Probabilidad de cruzamiento = 0.85
- Probabilidad de mutación = 0.01
- Número de generaciones a ejecutarse = 100

Tabla 3.2. Resultados obtenidos por el Algoritmo Genético

	Variante 1		**Variante 2**		**Variante 3**	
	Número de Componentes	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**
Comp30.fr	11	5813	13	14375	10	18672
Comp50.fr	18	14093	16	39953	15	35452
Comp70.fr	22	30500	20	38500	19	54188
Comp100.fr	28	38704	28	76359	26	95359
Comp120.fr	34	31297	35	102829	32	99608
Comp150.fr	41	54032	38	135250	39	79578
Comp170.fr	45	119531	47	232672	43	229172
Comp200.fr	61	53344	57	150781	55	162375
Comp220.fr	62	91531	60	225906	68	243469
Comp250.fr	81	103969	79	239031	74	272000
Comp270.fr	86	112172	83	356098	80	466938
Comp300.fr	93	235953	94	438761	91	543092

Se puede notar que los mejores resultados en este caso están dados por la tercera variante, es decir, la que maneja un mayor número de componentes en cada generación, el tiempo consumido por la misma es más alto que el empleado por el resto, pero la calidad de las soluciones se impone en la mayoría de los casos (ver Fig 3.2).

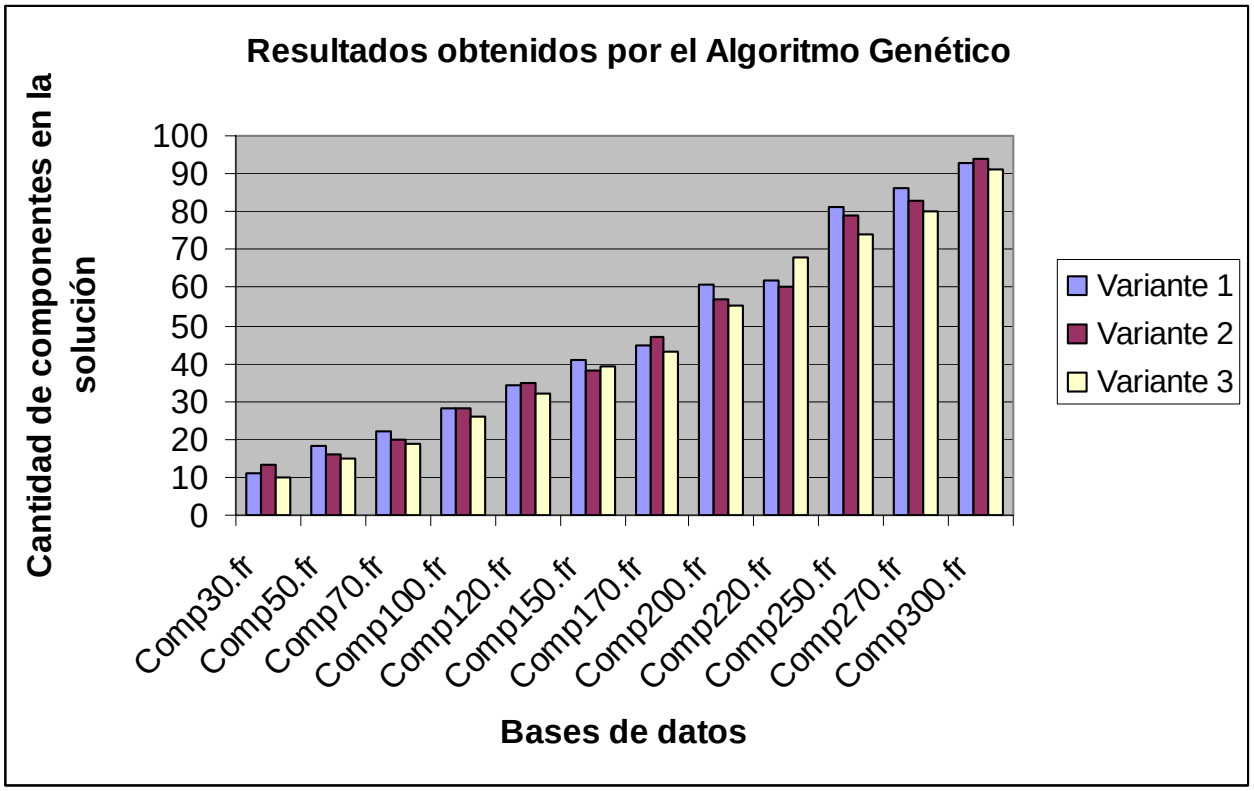


Figura 3.2. Resultados obtenidos por el Algoritmo Genético

3.2.3 Resultados obtenidos por el algoritmo Sistema de Colonias de Hormigas

En la aplicación de la optimización basada en colonias de hormigas también es necesario establecer los valores que se asignarán a los distintos parámetros del algoritmo.

En este caso los valores empleados para la evaporación de la feromona fueron $p,\varphi=0.1$. Para todas las pruebas se hizo uso de la variable $\alpha=1$ en la ecuación (1.5).

En el caso de los parámetros β y q_0, hay que ser cuidadosos al asignarle valores, pues son parámetros que guardan relación entre si. Es importante recordar que q_0 es el parámetro que incide en la decisión de movimiento de la hormiga (ecuaciones 1.4 y 1.5), mientras mayor es su valor, más probabilidad hay de que la hormiga explote el espacio de búsqueda, es decir, que se mueva hacia el

mejor nodo; en el caso de β, es quien le da peso al valor de la feromona en el movimiento de la hormiga (ecuaciones 1.4 y 1.5).

Un estudio realizado en la aplicación del algoritmo al problema del viajero vendedor demostró que mientras se aumentaba el valor de q_0, debía disminuirse el valor de β para obtener buenos resultados; es decir, que a medida que se va aumentando el nivel de explotación del problema se le debe ir dando menor peso en la decisión de movimiento a la feromona.

Este estudio se realizó para los valores β=1, 3, 5 y q_0 =0.3, 0.6, 0.9 y los mejores resultados se obtuvieron para la combinación q_0=0.9, β=3, la tabla en su totalidad se muestra en el anexo 1.

Basado en esto, se decidió analizar el comportamiento de estos parámetros para nuestro problema, teniendo en cuenta 3 variantes fundamentales, que fueron las que mejores resultados mostraron para el problema del Viajero Vendedor [49].

Variante 1: q_0=0.3, β=5

Variante 2:q_0=0.6, β=5

Variante 3: q_0=0.9, β=3

En todos los casos se utilizaron 10 hormigas y una cantidad de ciclos *nc=1000*.

Tabla 3.3. Resultados obtenidos por el algoritmo Sistema de Colonias de Hormigas

	Variante 1		**Variante 2**		**Variante 3**	
	Número de Componentes	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**
Comp30.fr	10	263780	10	235131	8	202750
Comp50.fr	12	527582	12	458930	10	247419
Comp70.fr	10	460162	8	233475	8	176357
Comp100.fr	17	534231	13	440855	15	356311
Comp120.fr	13	423105	12	372871	12	332357
Comp150.fr	18	645367	16	547715	15	416294
Comp170.fr	15	490134	15	426781	14	403464
Comp200.fr	22	503245	20	427683	21	415647
Comp220.fr	28	783412	25	639542	23	562319
Comp250.fr	28	790141	24	640127	26	645098
Comp270.fr	29	785094	29	754906	28	723631
Comp300.fr	35	762871	32	758875	32	753917

Al comparar los resultados obtenidos por las distintas variantes aplicadas, se puede apreciar que la tercera es la que mejores soluciones obtiene, tanto en calidad de la solución como en tiempo empleado en encontrar la misma, solo para 3 bases de datos el mejor resultado se obtuvo con la variante 2, y esto haciendo uso de un mayor tiempo de ejecución. Es decir, que para dar solución al problema en cuestión, es más factible el uso de un valor alto de $q_{0,}$ o lo que es lo mismo, mayor posibilidad de explotación del espacio de búsqueda. El siguiente gráfico permite observar estos resultados con mayor claridad.

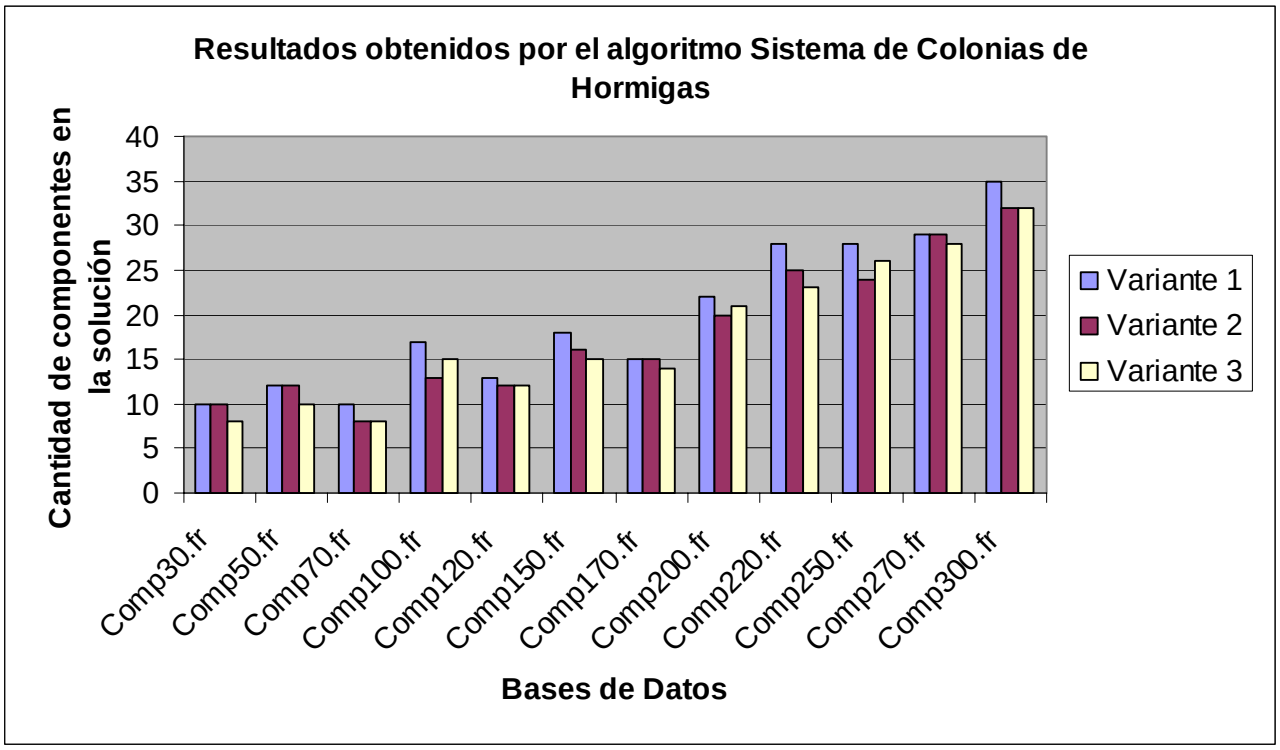


Figura 3.3. Resultados obtenidos por Sistema de Colonias de Hormigas

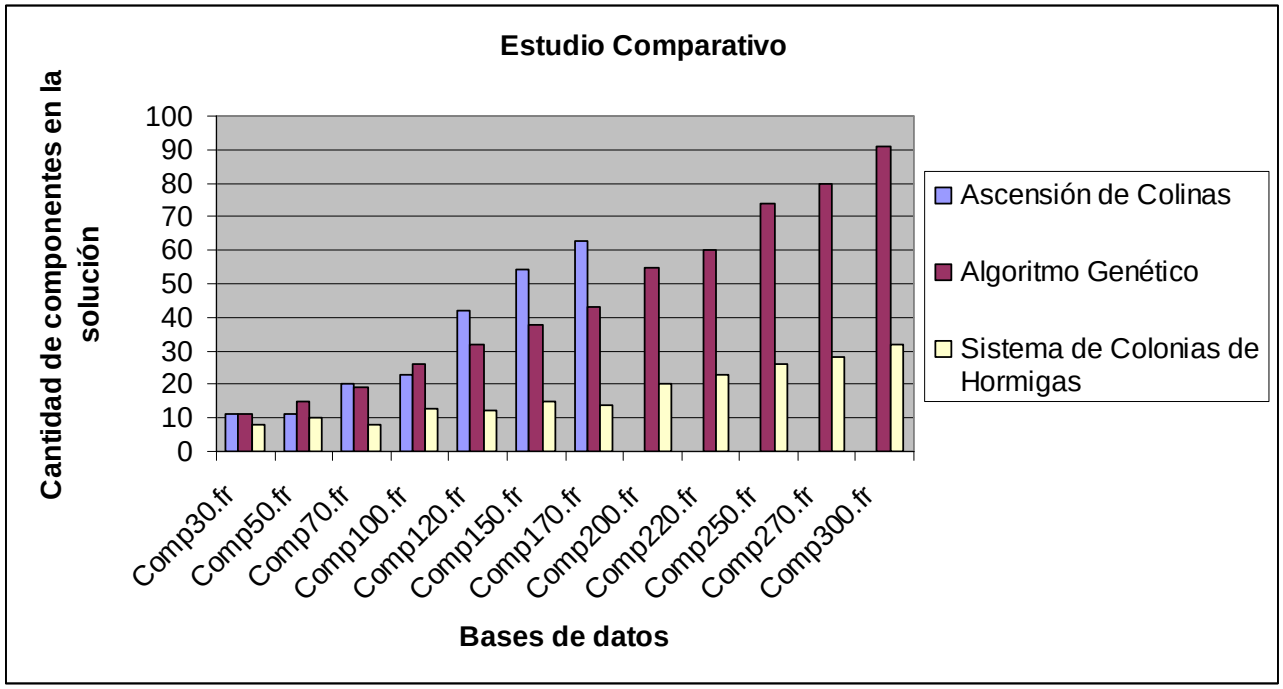
3.3 Estudio Comparativo

Teniendo en cuenta los resultados obtenidos por las distintas variantes en la aplicación de cada uno de los algoritmos, se realizó un estudio comparativo con las mejores soluciones reportadas por cada uno de ellos. Se realizaron 10 corridas de cada alternativa y su resultado fue promediado.

En el caso de la optimización basada en colonias de hormigas, los mejores resultados se alcanzaron con la variante 3, es decir, con valores de q_0=0.9 y β=3. Para el Algoritmo genético la variante tres, que manejaba una mayor cantidad de individuos resultó la mejor, mientras que en ascensión de colinas brindó mejores resultados la variante de mayor número de iteraciones.

Tabla 3.4. Estudio comparativo entre los algoritmos

	Ascensión de Colinas		**Algoritmo Genético**		**Sistema de colonias de hormigas**	
	Número de Componentes	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**	**Número de Componentes**	**Tiempo (ms)**
Comp30.fr	11	18656	11	5813	8	202750
Comp50.fr	11	18656	15	35452	10	247419
Comp70.fr	20	29172	19	54188	8	176357
Comp100.fr	23	16754	26	95359	13	440855
Comp120.fr	42	168469	32	99608	12	332357
Comp150.fr	54	271438	38	135250	15	416294
Comp170.fr	63	386719	43	229172	14	403464
Comp200.fr	-	-	55	162375	20	427683
Comp220.fr	-	-	60	225906	23	562319
Comp250.fr	-	-	74	272000	26	645098
Comp270.fr	-	-	80	466938	28	723631
Comp300.fr	-	-	91	543092	32	853917



3.4 Conclusiones parciales

En la ejecución de los distintos algoritmos es importante tener en cuenta el valor que se le asignará a los parámetros correspondientes al método heurístico empleado, pues en todos los casos estos tienen una gran influencia en el resultado que se obtendrá posteriormente. Para cada implementación se ejecutaron distintas variantes de conjugación de parámetros, según lo reportado en la bibliografía como lo sugerido para el método en cuestión, en aras de encontrar la variante que mejores resultados reportara para el problema abordado.

Para la implementación del algoritmo basado en el método Ascensión de Colinas se tuvo en cuenta variar la cantidad de iteraciones, probándose con dos variantes, donde la mayor cantidad de iteraciones obtuvo los mejores resultados.

El algoritmo genético implementado fue probado con 3 variantes en las cuales se varió el valor de los parámetros: número de generaciones y tamaño de la población, resultando mejor la opción en que se trabaja con mayor número de individuos.

En el caso del Sistema de Colonias de Hormigas, dos de los parámetros de mayor incidencia en los resultados alcanzados son q_0 y ß y el estudio presentado en este capítulo sobre los mismos, muestra la forma de escoger los valores para obtener buenos resultados. En este caso la combinación q_0=0.9 y β=3 reportó la mayor calidad en las soluciones.

El estudio comparativo realizado arrojó que las mejores soluciones son encontradas en todos los casos por el algoritmo Sistema Basado en Colonias de Hormigas, el cual también es el que mayor tiempo invierte en encontrar las soluciones en base a los parámetros seleccionados.

Conclusiones

Como resultado de esta investigación se desarrollaron tres algoritmos basados en métodos heurísticos, los cuales permiten a los investigadores en el campo de la configuración y del razonamiento funcional contar con nuevas vías para enfrentar la búsqueda de soluciones a partir de bases de datos de componentes, cumpliéndose de esta forma el objetivo general planteado, ya que:

- Se realizó un profundo análisis sobre los modelos de Razonamiento Funcional existentes, el cual evidenció que existen diversos estudios acerca de teorías y modelos de razonamiento funcional para problemas de diseño, donde las características del problema, que es minimizar la cantidad de componentes que aparecen en la solución, hace necesario el uso de métodos heurísticos cuando la complejidad del problema aumenta.
- Existe un creciente interés por el uso de los algoritmos bioinspirados para la solución de problemas de optimización, es por esto que se analiza la aplicabilidad de los métodos Ascensión de Colinas, Algoritmos Genéticos y Sistema de Colonias de Hormigas en la solución de problemas de configuración.
- Se proponen tres algoritmos basados en los métodos heurísticos antes mencionados, los cuales fueron modelados de acuerdo a las características del problema a resolver.
- Se realizó un análisis de los parámetros a utilizar en cada caso, analizando distintas variantes de ejecución según lo recomendado en la bibliografía para cada método empleado en aras de encontrar la variante que mejores resultados reportara para el problema abordado.
- El estudio comparativo realizado con las distintas soluciones encontradas por cada uno de los algoritmos propuestos arrojó que los mejores resultados son obtenidos en todos los casos por el algoritmo Sistema de Colonias de Hormigas, perteneciente a la metaheurística Optimización basada en Colonias de Hormigas.

Recomendaciones

El estudio realizado no agota este campo de investigación, de modo que existen diversas alternativas de trabajo científico posterior relacionado con esta problemática, entre las cuales se destacan:

- Analizar el efecto de dividir el proceso de búsqueda desarrollado por las hormigas en dos etapas, utilizando el modelo Optimización Basada en Colonias de Hormigas en Dos Pasos (*Two-Step Ant Colony Optimization, TS-ACO*) propuesto por el Laboratorio de Inteligencia Artificial de nuestro centro, el cual ha sido satisfactoriamente aplicado en Selección de Rasgos [50], el problema del Viajero Vendedor [51] y en problemas de Scheduling [52].
- Aplicar otros algoritmos de la metaheurística Optimización Basada en Colonias de Hormigas, por ejemplo, el Sistema de Hormigas Max-Min (*Max-Min Ant System*).
- Aplicar algoritmos de Estimación de Distribuciones, los cuales se encuentran dentro de la llamada computación evolutiva, los cuales han sido objeto de atención por parte de la comunidad científica en esta área.

Referencias bibliográficas

1. Far, B.H., *Functional Reasoning, Explanation and Analysis.* JAERI-M, Japan Atomic Energy Research Institute, Tokai, Japan,, 1992: p. 91-225.
2. B. H. Far and A. Halim Elamy, *Functional reasoning theories: Problems and perspectives.* Artificial Intelligence Engineering Design Anal. Manuf., 2005. **19**: p. 75-88.
3. Mary Lou Maher and Hsien-Hui, *Co-Evolution as a Computational and Cognitive Model of Design.* Research in Engineering Design, 2003. **14**: p. 47-64.
4. Peter Freeman and Allen Newell, *A Model for Functional Reasoning in Design*, in *IJCAI*. 1971.
5. Sanjay Mittal and Felix Frayman, *Towards a generic model of configuration tasks.* Proceedings of the 11th International Joint Conference on Artificial Intelligence -IJCAI '89, 1989.
6. Yoshikawa, H., *General design theory and a CAD system.* IFIP Man-machine communication in CAD/CAM, 1981: p. 35-88.
7. Yoshikawa, H., *Design theory for CAD/CAM integration.* Annals of the CIRP, 1985. **34**: p. 173-178.
8. G. Pahl and W. Beitz, *Engineering Design: A Systematic Approach.* Edited by Ken Wallace, Springer-Verlag, The Design Council, 1988.
9. A. Chakrabarti and P. Bligh, *A scheme for functional reasoning in conceptual design.* Design Studies, 2001. **22**: p. 493-517.
10. Yasushi Umeda and Tetsuo Tomiyama, *Functional Reasoning in Design.* IEEE Expert: Intelligent Systems and Their Applications, 1997. **12**: p. 42-48.
11. A.G. Cohn and JR Thomas, ed. *Artificial Intelligence and its applications*. 1986, John Wiley \& Sons, Inc.
12. E. Feigenbaum and J. Feldman, *Computers and Thought.* New York, McGraw-Hill., 1963.
13. A. Díaz, F.G., H.M. Ghaziri, J.L. Gonzalez, M. Laguna, P. Moscato and Tseng, and F.T., *Optimización Heurística y Redes Neuronales*. 1996, Madrid: Paraninfo.
14. Glover, F., *Future Paths for Integer Programming and Links to Artifical Intelligence.* Computers and Operations Research, 1986. **13**: p. 533–549.
15. Reeves, C.R., *Modern Heuristic Techniques for Combinatorial Problems*. 1995, UK: Ed. McGraw-Hill.
16. Kelly, I.H.O.a.J.P., *Meta-Heuristics: Theory and Applications*. 1996, Boston: Ed. Kluwer Academic.
17. Glover, F., *Heuristics for Integer Programming Using Surrogate Constraints.* Decision Sciences, 1977. **8**: p. 156-166.

18. S. Kirkpatrick, D.G.a.M.P.V., *Optimization by Simulated Annealing.* Science,, 1983. **220** (4598): p. 671-680.
19. Goldberg, D.E., *Genetic Algorithms in Search*. Optimization and Machine Learning. 1998, University of Alabama: Addison-Wesley Publishing Company.
20. Eberhart, J.K.a.R.C. *Particle swarm optimization.* in *on neural networks*. 1995. Piscataway, NJ.
21. Dorigo M. and LM. Gambardella, *Ant Colonies for the Traveling Salesman Problem.* BioSystems, 1997. **43**: p. 73-81.
22. M. Dorigo and T. Stützle, *ACO Algorithms for the Traveling Salesman Problem*. Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications. 1999, EEUU: John Wiley & Sons.
23. Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach, Chapter 4*. 1995: Prentice-Hall.
24. Morales, E. *Búsqueda, Optimización y Aprendizaje*. 2004 [cited.
25. Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. 1989: Addison-Wesley.
26. MICHALEWICZ, Z., *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag. Second, Extended Edition, New York,USA, 1994.
27. Holland J. H., *Adaptation in Natural and Artificial Systems. An introductory analysis with applications to Biology, Control and Artificial Intelligence.* University of Michigan Press, 1975.
28. Bennett, K., Ferris, M. C., and Ioannidis, Y. E,, *A Genetic Algorithm for Database Query Optimization.* New York: John Wiley, Chichester, 1996.
29. Booker, L.B., *Intelligence behavior as an Adaptation to the Tusk Environment.* University of Michigan, 1982.
30. Jong, K.A.D., *Genetic Algorithms: a 10 years perspective.* 1970: p. pp.169-177.
31. Michalewicz Z., K., J., Kazemi, M., Janikow, C,. *Genetic Algoritms and Optimal Control Problems*. in *29th IEEE Conference on Decision and Control*. 1990. Honolulu.
32. Almirón, M., *Ant System*. 2000, Universidad Nacional de Asunción.
33. Holland, J., *Adaptation in Natural and Artificial Systems.* MIT Press, 1992.
34. Jiménez, F. *Vida Artificial*. 1999 [cited; Available from: http://complex.us.es/~jimenez/CA/ac/node19.html.
35. Liekens, A. *Alife Online*. 2000 [cited; Available from: http://alife.org/.
36. Paolo, E.D. *Artificial Life Bibliography of On-line Publications*. 2000 [cited; Available from: http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/alife.html.
37. M. Dorigo and G. Di Caro, *The Ant Colony Optimization meta-heuristic*, in *New Ideas in Optimization*, M. Hill, Editor. 1999: London UK. p. 11-32.

38. M. Dorigo, G.D.C., and L. M. Gambardella,, *Ant algorithms for discrete optimization.* Artificial Life, 1999. **5**(2): p. 137-172.

39. M. Dorigo, V.M., and A. Colorni, *The Ant System: Optimization by a colony of cooperating agents.* IEEE Transactions on Systems Man, and Cybernetics, 1996. **26**: p. 29-41.

40. M. Dorigo and L. M. Gambardella, *Ant Colony System: A cooperative learning approach to the traveling salesman problem.* IEEE Transactions on Evolutionary Computation, 1997. **1**(1): p. 53-66.

41. T. Stützle and H. Hoos, *MAX-MIN Ant System.* Future Generation Computer Systems, 2000. **16**(8): p. 889-914.

42. B. Bullnheimer, R.F.H.a.C.S., *A new rank-based version of the Ant System: A computational study.* Central European Journal for Operations Research and Economics, 1999. **7**(1): p. 25-38.

43. O. Cordón, I.F.d.V., F. Herrera, and L. Moreno,. *New ACO model integrating evolutionary computation concepts: The Best-Worst Ant System*. in *ANTS 2000*. 2000. Université Libre de Bruxelles, Belgium,.

44. L. Gambardella and M. Dorigo, *HAS-SOP: An Hybrid Ant System for the Sequential Ordering Problem*, in *Technical Report IDSIA11-97*. 1997: Lugano - Suiza.

45. B. Bullnheimer, R.H.a.C.S. *An improved ant system algorithm for the vehicle routing problem*. in *Annals of Operations Research*. 1999: Nonlinear Economic Dynamics and Control.

46. V. Maniezzo and A. Colorni, *The Ant System applied to the Quadratic Assignment Problem*, in *Transactions on Knowdledge and Data Engineering*. 1999, IEEE Estados Unidos.

47. G. Di Caro and M. Dorigo. *Mobile Agents for Adaptive Routing*. in *Proceedings of the 31st Hawaii International Conference on System*. 1998. Hawaii.

48. Mitchel, M., *An Introduction to Genetic Algoritms.* MIT Third Printing., 1997.

49. A. Puris, R.B., Y. Martínez. *Aplicación de la optimización basada en colonias de hormigas al problema del viajero vendedor*. in *V Conferencia Internacional de Ciencias Empresariales*. 2006. Cuba.

50. A. Puris, R.B., A. Nowe, M. Garcia, Y. Martínez,, *Two Step Ant Colony System to Solve the Feature Selection Problem.* Lecture Notes in Computer Science, 2006. **4225**: p. 588-596.

51. A. Puris, R.B., A. Nowe, Y. Martínez,, *Two-Stage Ant Colony System for solving the Traveling Salesman Problem*, in *Research in Computer Science*. 2006.

52. A. Puris, R.B., A. Nowe, M. Garcia, Y. Martínez,. *Two-Stage ACO to solve the Job Shop Scheduling Problem*. in *12th Iberoamerican Congress on Pattern Recognition CIARP 2007*. 2007. Chile.

Anexos

Anexo 1. Estudio experimental para los parámetros β y q_0 en la aplicación de la optimización basada en colonias de hormigas

Base de datos	**N**	**Mejor Solución**	**Valor q_0**	**Valor β**	**Solución ACS**
bays29.tsp	29	2020	0.3	1	2595
				3	2147
				5	**2057.66667**
			0.6	1	2141.66667
				3	2102.33333
				5	**2072.66667**
			0.9	1	2104.66667
				3	**2052,66667**
				5	2103.33333
gr24.tsp	24	1272	0.3	1	1512.333333
				3	1325
				5	**1310.666667**
			0.6	1	1366
				3	**1300**
				5	1319.666667
			0.9	1	**1301**
				3	1339
				5	1349.333333
rd100.tsp	100	7910	0.3	1	16145.66667
				3	9628
				5	**8765.666667**
			0.6	1	11588.66667
				3	8926
				5	**8716.666667**
			0.9	1	9030
				3	**8850.333333**
				5	8872.333333

gr120.tsp	120	6942	0.3	1	14259.66667
				3	9181.333333
				5	**8254.333333**
			0.6	1	10020.33333
				3	7946.333333
				5	**7659.333333**
			0.9	1	7550.333333
				3	**7443.666667**
				5	7501.333333
st70.tsp	70	675	0.3	1	1286.333333
				3	841
				5	**786.3333333**
			0.6	1	957.6666667
				3	802
				5	**789**
			0.9	1	797.6666667
				3	**786.3333333**
				5	798
tsp225.tsp	70	3919	0.3	1	9504
				3	5221
				5	**4578**
			0.6	1	6315.666667
				3	4682.666667
				5	**4417**
			0.9	1	4574.666667
				3	**4322**
				5	4328