UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"DOS ENFOQUES PARA RESOLVER EL PROBLEMA DE LA SUPERSECUENCIA COMÚN MÁS CORTA: PROGRAMACIÓN DINÁMICA Y HEURÍSTICA ACO"

TESIS
PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

PRESENTA:
ERIC PÉREZ HERNÁNDEZ

DIRECTOR DE TESIS:
M.C. MARCELA RIVERA MARTÍNEZ

HUAJUAPAN DE LEÓN, OAXACA. MARZO DE 2006

A mis padres, Baltazar e Hilda

Gracias por su amor, apoyo y confianza que
me han brindado a lo largo de mi vida

A mis hermanos, Alejandro y Laura

Gracias por apoyarme en todo momento
y estar siempre conmigo

A mi abuelita, Josefa†

Por ser ejemplo de sacrificio, lucha y trabajo

A todos mis tíos y primos,

Muchas gracias por todo su apoyo incondicional.

Agradecimientos

Agradezco de forma especial a la M.C. Marcela Rivera Martínez por brindarme su apoyo incondicional en el tiempo en que desarrollé mi tesis, demostrándome su paciencia y amistad.

Al M.C. Luis René Marcial Castillo, a la M.C. Verónica Rodríguez López y al Ing. Hugo Enrique Martínez Cortés, por todo el apoyo que me brindaron en la revisión de esta tesis y por sus consejos.

A mis compañeros, gracias por su eterna amistad.

A mis profesores, por su dedicación y tiempo empleados para compartir sus conocimientos conmigo.

Índice general

Índice de figuras

Introducción

Un problema computacional que se presenta al tratar de resolver problemas de optimización combinatoria, es la explosión combinatoria, la cual consiste en que hay un número muy grande de soluciones posibles para dicho problema. Para lo cual, si se tienen que examinar todas o la mayoría de éstas, toma mucho tiempo el poder hacerlo.

En algunos problemas, donde el espacio de búsqueda es suficientemente pequeño y enumerable, se pueden usar métodos de búsqueda exhaustiva o enumerativa, en donde se examinan todas las soluciones posibles, y se toma como buena aquella que tenga el máximo (mínimo) valor de la función objetivo, en caso de que se trate de un problema de maximización (minimización).

El problema con la búsqueda exhaustiva no sólo es que a menudo es infactible, sino que tiene un mal comportamiento con respecto al aumento del tamaño del espacio de búsqueda; y el hecho de que necesite una cantidad de memoria de la computadora proporcional al tamaño del problema, hace que no se pueda aplicar en la mayor parte de los casos[1].

Algunos problemas clásicos en donde se presenta la explosión combinatoria, al tratar de obtener todas las alternativas posibles para su solución por medio de la computadora, resulta poco factible. Por ejemplo, en el problema del agente viajero, en que dado un grafo, se tiene que salir de una ciudad y regresar a la misma, después de haber visitado todas las demás ciudades pasando una sola vez por cada una de ellas, con el costo mínimo de la distancia recorrida del viaje, si se tienen n ciudades en total que recorrer, entonces, existen $(n-1)!$ soluciones factibles, por lo que si una computadora fuera programada para examinar soluciones a razón de un billón de soluciones por segundo; tendríamos que para n igual a 25 ciudades, el obtener una solución óptima, tardaría alrededor de 19,674 años[2].

Existen muchos problemas combinatorios de optimización, para los cuales es impráctico encontrar una solución óptima. En estos problemas, el único camino razonable es utilizar un

algoritmo heurístico que produzca de manera rápida buenas soluciones, aunque éstas no sean necesariamente las óptimas.

Uno de estos problemas, es el de la Supersecuencia Común más Corta (SCS), que consiste en encontrar una cadena que cumpla con lo siguiente: que sea la de menor longitud y que contenga todas las cadenas dadas de un lenguaje, formadas de un mismo alfabeto.

El problema de la Supersecuencia Común más Corta tiene aplicaciones en la inteligencia artificial (planeación), en ingeniería mecánica, compresión de datos, etc., y es un problema NP-duro bajo varias restricciones concernientes al tamaño del alfabeto y la longitud de las cadenas[3].

En este trabajo, se plantea desarrollar una aplicación basada en el uso de una heurística ACO (Optimización basada en Colonias de Hormigas), este algoritmo está inspirado en la observación de colonias de hormigas reales, las cuales son insectos sociales y tienen un comportamiento más dirigido a la supervivencia de la colonia que a un individuo en particular. Las hormigas cooperan indirectamente al cambiar la representación del problema, de tal forma que las siguientes hormigas son guiadas hacia buenas soluciones.

Al no contar con problemas reales para poder realizar pruebas con el software desarrollado, se generarán problemas sintéticos. Debido a que el problema está clasificado como de complejidad NP, las instancias para las cuales se probará el problema de la SCS serán "pequeñas".

Además de la aplicación basada en una heurística ACO, se instrumenta un algoritmo basado en una técnica de Programación Dinámica a fin de poder comparar los resultados obtenidos por ambas técnicas.

La Programación Dinámica se aplica a problemas de optimización, en donde existen muchas soluciones posibles, donde cada solución tiene un valor y se desea encontrar una solución con el valor óptimo (ya sea máximo o mínimo). El algoritmo de Programación Dinámica divide el problema original en subproblemas, y resuelve un subproblema una vez, y entonces guarda la respuesta en una tabla, evitando el trabajo de volver a calcular la respuesta cada vez que el subproblema es encontrado.

Existen muchos problemas para los cuales no sólo deseamos encontrar el valor de la solución óptima, sino que además deseamos conocer la composición de esta solución, es decir, los elementos que forman parte de ella. En estos casos es necesario ir conservando no sólo los valores de las

soluciones parciales, sino también cómo se llega a ellas. Esta información puede ser almacenada en la misma tabla donde se guardan las soluciones parciales, o bien en otra tabla.

De esta manera, los objetivos de este trabajo son:

- Estudiar sistemas multi-agente, en los cuales el comportamiento de cada agente -hormiga artificial-, está inspirado en el comportamiento real de las hormigas.
- Instrumentar un software que resuelva el problema SCS mediante un algoritmo basado en colonias de hormigas.
- Instrumentar un software basado en la técnica de Programación Dinámica para resolver el problema SCS.
- Comparar los resultados obtenidos por ambas técnicas: Heurística ACO y Programación Dinámica, para resolver el problema SCS.

Este trabajo se divide en seis capítulos, en el primer capítulo se define el problema a resolver, así como conceptos básicos referentes al problema SCS y los de complejidad computacional.

En el capítulo dos se presenta la descripción de las heurísticas y metaheurísticas, además de su clasificación. También se explica el funcionamiento general de los algoritmos basados en optimización de colonias de hormigas.

La descripción de la técnica de Programación Dinámica, así como algunas de sus aplicaciones se presentan en el tercer capítulo.

El capítulo cuatro consta de los algoritmos a instrumentar, tanto el de Programación Dinámica como el de colonias de hormigas.

Las pruebas y resultados obtenidos por las aplicaciones desarrolladas se muestran en el quinto capítulo.

Las conclusiones a las que se llegaron al finalizar el trabajo son expuestas en el capítulo seis.

Se adicionan además, la bibliografía utilizada durante el desarrollo de éste trabajo, así como apéndices que contienen el manual de usuario correspondiente a la aplicación creada, ejemplos de prueba, formato del archivo de entrada y salida y finalmente un glosario de términos técnicos utilizados en este documento.

Capítulo 1

Conceptos Básicos

En este capítulo, se presentan los conceptos básicos que son necesarios para conocer y comprender el tema tratado en esta tesis, que es la solución del problema de la Supersecuencia Común más Corta[4].

Definición 1 ***Símbolo.***

Es una entidad abstracta. Las letras y los dígitos son ejemplos de símbolos usados con frecuencia.

Definición 2 ***Cadena o Palabra.***

Es una secuencia finita de símbolos yuxtapuestos. Por ejemplo, *a, b* y *c* son símbolos y *abcb* es una cadena.

Definición 3 ***Longitud de una cadena w.***

Se denota como $|w|$, es el número de símbolos que componen la cadena.

Ejemplo: $w = abcb$ tiene longitud cuatro, es decir, $|w| = 4$.

La cadena vacía denotada por $\in$, es la cadena que consiste de cero símbolos. Por lo tanto, $|\in| = 0$.

Definición 4 ***Alfabeto.***

Es un conjunto finito de símbolos, y se denota mediante Σ.

$$\text{Ejemplo: } \Sigma = \{0, 1\}\,,\ \Sigma = \{a\}\,.$$

Definición 5 ***Lenguaje.***

Es un conjunto de cadenas tomadas de algún alfabeto. El conjunto vacío, ϕ, y el conjunto formado por la cadena vacía $\{\in\}$ son lenguajes. Nótese que son diferentes: el último tiene un elemento mientras que el primero no.

$$\text{Ejemplo: } L = \ \{aa, ba, ab, bb\}\ \text{, bajo el alfabeto } \Sigma = \{a, b\}\,.$$

Definición 6 ***Supersecuencia.***

Se dice que S es una supersecuencia de una cadena T, si S se puede obtener de T, mediante la inserción de cero o más símbolos.

Ejemplo: Dado

$$\Sigma = \{a, b, c\} \qquad\qquad y$$
$$L\ = \{ab, ba, ac, abc\}\,.$$

Una supersecuencia es:

$S\ =$ *abbaacabc*, que es la unión de todas las cadenas del lenguaje L.

Definición 7 ***Supersecuencia Común más Corta.***

Dado un lenguaje (L) de cadenas bajo un alfabeto (Σ), una supersecuencia común más corta (S) de L es una cadena con la menor longitud, que es una supersecuencia de cada cadena en L.

Ejemplo: Dado

$$\Sigma = \{a, b\} \qquad y$$
$$L\ = \{ab, ba\}\,.$$

Supersecuencias de $L = \{abba, aba, baab, bab\}$

La Supersecuencia Común más Corta es:

$S = aba$, o también $S = bab$.

$|S| = 3$.

1.1. Complejidad Computacional.

Muchos problemas pueden plantearse como problemas de decisión, en donde basta con dar una respuesta afirmativa o negativa a una instancia del problema dado. Y generalmente, la teoría de la complejidad busca replantear los problemas como problemas de decisión.

La Teoría de la Complejidad Computacional es la parte de la Teoría de la Computación que clasifica los algoritmos como buenos o malos y los clasifica de acuerdo a la dificultad para resolverlos. Además, estudia los recursos requeridos durante el cálculo para resolver un problema. Los recursos comúnmente estudiados son el tiempo (número de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (cantidad de memoria utilizada para resolver un problema). Se pueden estudiar igualmente otros parámetros, tales como el número de procesadores necesarios para resolver el problema en paralelo.

Los problemas que tienen una solución con orden de complejidad lineal, son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño.

Hoy en día, las máquinas resuelven problemas mediante algoritmos que tienen como máximo una complejidad o costo computacional polinomial, es decir, la relación entre el tamaño del problema y su tiempo de ejecución es polinomial. Éstos son problemas agrupados en el conjunto P. Los problemas con costo factorial o combinatorio están agrupados en NP. Estos problemas no pueden ser resueltos por una máquina en un tiempo razonable.

1.2. Clases de complejidad.

A continuación se definen las clases de complejidades de los problemas[5,6].

1.2.1. Clase de complejidad P.

Es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial, lo que corresponde intuitivamente a problemas que

pueden ser resueltos aún en el peor de sus casos. Por ejemplo, el problema de la búsqueda binaria, algoritmos de ordenamiento: quicksort, burbuja, etc.

1.2.2. Clase de complejidad NP.

Es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina de Turing no determinista en tiempo polinomial. Esta clase contiene muchos problemas que se desean resolver en la práctica, incluyendo el problema de satisfacibilidad booleana, el problema del camino Hamiltoneano y el problema de la cobertura de vértices, entre otros.

1.2.3. Clase de complejidad NP-Completo.

Son problemas NP más difíciles de resolver.

Definición 8 ***NP-Completo.***

Un Problema X esta en la clase NP-Completo si:

a)$X \in NP$, y
b)$\forall\ X' \in NP, X'$ es transformado (reducible) polinomialmente a X.

Definición 9 ***Reducibilidad.***

Se dice que un lenguaje L_1 es reducible en tiempo polinomial a un lenguaje L_2 si hay una función de cadena computable en tiempo polinomial para la cual $f(u) \in L_2$ sí y sólo sí $u \in L$.

Algunos problemas que están en NP-Completo son: agente viajero, coloración de mapas y coloración de grafos.

1.2.4. Clase de complejidad NP-Duro.

Son los problemas que pueden o no estar en NP pero que son al menos tan difíciles como los NP-Completo.

Definición 10 ***NP-Duro.***

Si $\forall\ X' \in NP, X'$ es transformado polinomialmente a X, pero no necesariamente $X \in NP$, decimos que X es NP-Duro.

Las versiones de optimización de un problema NP-Completo son NP-duro. Un ejemplo de esta clase, es el problema de paro, que consiste en tomar un programa y sus datos y decidir si va a terminar o si se ejecutará indefinidamente.

Capítulo 2

Heurísticas y Metaheurísticas

En éste capítulo, se explican algunos métodos para la solución de problemas de optimización combinatorios, así como su clasificación y descripción de cada uno de ellos.

Muchos problemas de optimización no pueden ser resueltos utilizando métodos exactos, ya sea, porque tienen un alto grado combinatorio o por la dificultad de generar un modelo basado en programación matemática que represente exactamente una situación real. Para este tipo de problemas, se han venido generando desde la década de los sesenta métodos conocidos como heurísticas, capaces de encontrar soluciones de buena calidad, aproximadas a la solución óptima y en un periodo de tiempo razonable[3].

De acuerdo con el estándar ANSI/IEEE 100-1984, "la heurística trata de métodos o algoritmos exploratorios, durante la solución de problemas, en los cuales las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final."

Según George Polya, la base de la heurística está en la experiencia de resolver problemas y en ver cómo otros lo hacen.

Las causas que hacen que sea más factible el uso de los algoritmos heurísticos para obtener la solución de un problema, se puede resumir en los siguientes casos:

1. Cuando no existe un método exacto para solucionarlo o éste requiere mucho tiempo de cálculo o de memoria. Significa que es preferible tener una solución que sea aceptablemente buena, cercana a la óptima a no tener solución alguna.

2. Cuando no se necesita la solución óptima. Esto es, cuando el beneficio obtenido al tener

una solución óptima y el de una subóptima no sea muy significativo.

3. Cuando los datos son poco fiables. En este caso, si los datos de entrada no son exactos, no tendría sentido obtener una solución exacta, ya que sería una aproximación a la real.

4. Cuando existen limitaciones de tiempo y/o espacio de almacenamiento de los datos, lo que obliga al empleo de métodos que obtengan una respuesta rápida, aunque ésta no sea la óptima.

5. Como paso intermedio en la aplicación de otro algoritmo. En ocasiones, los resultados obtenidos por heurísticas, sirven como inicio para algoritmos exactos de tipo iterativo.

Dentro de las ventajas que se tienen al utilizar métodos heurísticos en lugar de métodos exactos, es que los primeros tienen una mayor flexibilidad para adecuarlo al problema. Otra ventaja, es que generalmente ofrecen más de una solución, permitiendo elegir la que más convenga, debido a que hay factores que no se toman en cuenta al hacer el modelo, ya que no son medibles o no se tiene esa información. Las heurísticas tienen la característica de ser más entendibles que las técnicas exactas que utilizan métodos matemáticos.

Sin embargo, las heurísticas también tienen inconvenientes, uno de ellos es que no se puede saber qué tan cercana está la solución obtenida a través de éstas con respecto a la óptima. Lo único que se conoce es que la solución subóptima obtenida por las heurísticas es menor o igual que la óptima, si es que se trata de un problema de maximización. Aunque existen métodos para realizar acotaciones al problema, como por ejemplo, relajar el problema, que consiste en quitar restricciones al problema para incrementar el espacio de solución y que sea más fácil resolver el problema, nada nos dice qué tan buena es la solución obtenida con respecto a la óptima.

De lo anterior, se puede decir que para verificar la eficiencia de la heurística, se necesitan resolver problemas y comparar los resultados obtenidos por métodos exactos si es posible, o con otras heurísticas de las cuales se conozcan sus resultados.

2.1. Tipos de Heurísticas.

Existen diversos tipos de heurísticas de acuerdo a la manera en la que buscan y construyen la solución. Una clasificación es la siguiente[7]:

1. Métodos constructivos. Los cuales añaden paulatinamente componentes individuales a la solución hasta que obtiene una que sea factible. Dentro de este tipo se encuentran los algoritmos golosos o devoradores (greedy), los cuales construyen la solución paso a paso, buscando el máximo beneficio en cada uno de ellos.

2. Métodos de descomposición. Los cuales dividen el problema original en subproblemas más pequeños, donde la salida de uno es la entrada de otro, de modo que al resolver cada parte se obtenga una solución global, es decir, divide y vencerás.

3. Métodos de reducción. Tratan de identificar alguna característica que deba de tener la solución óptima y así simplificar el problema.

4. Manipulación del modelo. Se manipula el modelo original para hacerlo más sencillo de resolver, como reducir el espacio de solución, reducir variables, eliminar restricciones, etc. y de la solución obtenida de éste, se deduce la solución óptima para el problema original.

5. Métodos de búsqueda por entornos. Dentro de éstos métodos se encuentran las meta-heurísticas. Parten de una solución factible inicial que pudo ser obtenida a través de otra metaheurística, y mediante modificaciones a ésta, se van obteniendo otras soluciones factibles de su "entorno", tomando como solución óptima a la mejor de todas, mientras no se cumpla una condición de paro.

En los inicios de las heurísticas, se generaron métodos orientados específicamente a la solución de cada problema, gran parte de estos métodos fueron generados inspirándose en la solución de problemas de fácil representación pero de muy difícil solución como lo son: el problema del agente viajero, el problema de la mochila, el problema de la cobertura de conjuntos, etc. Por la naturaleza diferente de estos problemas, los métodos que se generaron eran útiles apenas para el problema en el cual habían sido inspirados, a partir de la década de los 80's se ha generado una familia de métodos conocidos como meta-heurísticas que ahora tienen la capacidad de ser aplicables a problemas de diversa naturaleza, es decir, una misma meta-heurística puede ser utilizada para resolver problemas que provienen de diversos sectores[3].

2.2. Métodos Metaheurísticos.

Las metaheurísticas son una clase de métodos de aproximación, que se diseñan para atacar problemas difíciles para los cuales las heurísticas de propósito específico han fracasado en dar resultados efectivos y eficientes. Permiten crear nuevos híbridos al combinar diferentes conceptos derivados de las heurísticas clásicas, la inteligencia artificial, la evolución biológica, los sistemas neuronales, la mecánica estadística y el psicoanálisis freudiano[8].

2.2.1. Búsqueda Tabú.

Se basa en utilizar memoria de corto y largo plazo, para recorrer el espacio de soluciones. En cada etapa, el método genera la mejor solución perteneciente a una vecindad predefinida, y verificando el cumplimiento de ciertas restricciones de memoria y ciclaje, dirige la búsqueda hasta ésta nueva solución generada, a partir de la cual inicia una nueva etapa. Trabajando de esta manera, el algoritmo gradualmente mejora la calidad de la solución etapa a etapa, hasta que se detiene con la mejor solución encontrada durante este proceso de búsqueda; algunas veces esta es la solución óptima, en otros casos apenas es una solución aproximada a la solución óptima.

2.2.2. Recocido Simulado.

Este método se generó a partir de una analogía entre el fenómeno de enfriamiento de sustancias puras y un problema de optimización. El método realiza una búsqueda local al generar una solución vecina de manera aleatoria, si la solución es mejor que la solución actual, se actualiza ésta y se reinicia el proceso de búsqueda a partir de esta nueva solución, en caso contrario, si la solución vecina generada es peor que la solución actual se le da una oportunidad probabilística al método, para que acepte una solución de peor calidad que la de la solución actual. Este proceso de búsqueda se repite hasta que el algoritmo ya no es capaz de generar soluciones de mejor calidad que la actual.

2.2.3. Algoritmos Genéticos.

Los algoritmos genéticos son métodos de búsqueda que recorren el espacio de posibilidades en forma paralela y aleatoria, obedecen a una analogía con la evolución de las especies de Darwin. En cada etapa, se tiene una población de soluciones posibles para el problema, a partir de ésta, se genera una nueva población de soluciones mediante operadores que emulan la selección entre las especies del cruzamiento y la mutación. El método trabaja, generación tras generación, mejorando la calidad de la mejor solución de cada población hasta que algún criterio de detección se cumpla, por ejemplo, que el esfuerzo computacional que se ha invertido en la solución del problema ha superado el límite predefinido.

2.2.4. Redes Neuronales.

Las redes neuronales son modelos matemáticos que representan la interacción entre las neuronas que se encuentran en el cerebro humano. Su uso en optimización, se origina a partir de la idea de que la ejecución de un algoritmo de aprendizaje de una red neuronal, simultáneamente minimiza una función de energía en la red. Al tener un problema de optimización, se realiza un mapeo de la función objetivo del problema sobre la función de energía de la red y ejecutando el método de aprendizaje se obtiene simultáneamente la solución para un problema de optimización.

2.2.5. Métodos basados en Inteligencia Artificial.

Los métodos basados en inteligencia artificial, conocidos como métodos de búsqueda exhaustiva, generan de manera constructiva caminos hacia la solución óptima de un determinado problema, por ejemplo los métodos A* y AO*. Estos métodos se han mostrado muy eficientes en la solución de algunos tipos de problemas combinatorios.

2.2.6. Métodos Basados en Colonia de Hormigas.

Parten principalmente del estudio del comportamiento de las hormigas sociales, realizado principalmente por Pierre-Paul Grassé y Edgard O. Wilson, que vienen a entender la colonia de hormigas como un solo organismo, cuyos componentes u órganos son las hormigas en sí. Los

componentes de una colonia se comportan de tal forma que el comportamiento de la misma es mucho más que la suma de sus partes, o incluso de la inteligencia de cada una de las partes. Para explicar esto, Pierre Paul Grassé introdujo el concepto de stigmergia, que consiste en la asociación de varios órganos para la producción de un trabajo, por la comunicación que se realiza entre los miembros de la colonia a través de la modificación de forma indirecta del ambiente en el que se mueven. De esta forma se puede explicar la realización de obras colectivas sin necesidad de la intervención de una autoridad central. Estas ideas fueron usadas por investigadores como Jean-Louis Deneaubourg y Marco Dorigo.

El comportamiento de las hormigas en el rastro que dejan de feromonas y su seguimiento, ha sido estudiado por medio de experimentos controlados por varios investigadores. Uno de éstos, consiste en un puente doble que conecta un nido de hormigas y una fuente de comida; primeramente, la longitud de los dos puentes es del mismo tamaño, y las hormigas son libres de moverse por cualquiera de los dos caminos, ya que no existe ninguna cantidad de feromona depositada en ninguna ruta; al moverse, lo hacen aleatoriamente eligiendo uno u otro puente, hasta que después de un tiempo, todas las hormigas utilizan un mismo camino, debido a que hay mayor cantidad de feromona que en el otro, ésto hace que las hormigas lo elijan hasta que poco a poco todas vayan por ese puente.

En un segundo experimento, la longitud de uno de los caminos es lo doble del otro. Después de algún tiempo, se observa que las hormigas eligen el camino más corto. De igual manera que en el primer experimento, las hormigas salen del nido para explorar su ambiente y llegan a un punto donde tienen que decidir cual camino tomar, y al no haber ninguna cantidad de feromona depositada, eligen cualquiera aleatoriamente. Las hormigas que eligieron el camino más corto, llegan más rápido a la fuente de comida y comienzan su camino de regreso al nido, y depositan cierta cantidad de feromona en él; esto hace que se acumule mayor cantidad de feromona en el puente más corto que en el largo, y a pesar de que hay hormigas que se van por el camino más largo, con el paso del tiempo, todas utilizan el puente más corto.

Casi todos los comportamientos de una colonia de hormigas, desde la construcción de estructuras, hasta la localización de comida, se pueden usar para algún tipo de problema de búsqueda; toda esta familia de algoritmos se denominan Optimización por Colonia de Hormigas (Ant Colony Optimization).

A continuación se muestra el algoritmo ACO en pseudo-código:

procedimiento ACO_meta-heurística()
mientras(criterio_de_terminación_no_satisfecho)
planeación_de_actividades
generación_y_actividad_de_hormigas();
evaporación_de_feromona();
acciones_de_demonio(); {opcional}
fin planeación_de_actividades
fin mientras
fin procedimiento
procedimiento generación_y_actividad_de_hormigas()
mientras recursos disponibles()
planear_creación_de_nueva_hormiga();
nueva_hormiga_activa();
fin mientras
fin procedimiento
procedimiento nueva_hormiga_activa() {ciclo de vida de la hormiga}
inicializar_hormiga();
μ = actualizar_memoria_hormiga();
mientras (estado actual $\neq$ estado destino)
A = leer_tabla_de_ruteo_de_hormiga_actual();
P = computar_probabilidades_de_transición(A,μ, Ω);
siguiente_estado = aplicar_política_de_desición_de_hormiga(P,Ω);
mover_siguiente_estado(siguiente_estado);
si (actualización_de_feromona_en_linea_paso_a_paso)
depositar_feromona_en_arco_visitado();
actualizar_tabla_de_ruteo_de_hormiga();
fin si
μ =actualizar_estado_interno();
fin mientras

si (actualización_de_feromona_en_linea_retrasado)

para cada arco visitado $\in \psi$ **hacer**

depositar_feromona_en_arco_visitado();

actualizar_tabla_de_ruteo_de_hormiga();

fin para cada

fin si

hormiga_muere();

fin procedimiento

Además de la actividad de las hormigas, un algoritmo ACO incluye dos procedimientos: *acciones_de_demonio*(); que es opcional, y puede ser utilizado para realizar acciones que no puedan llevarse a cabo por hormigas individuales, como por ejemplo recolectar información global que puede ser útil para decidir si se deposita o no feromona adicional desde una perspectiva no local; y $evaporación\_de\_feromona()$; que se refiere a que la intensidad del rastro de la feromona en las conexiones automáticamente se decrementa con el tiempo. La evaporación de feromona es necesaria para evitar una convergencia rápida del algoritmo hacia una región sub-óptima del problema, es útil para favorecer la exploración de nuevas áreas del espacio de búsqueda.

2.3. Solución de SCS mediante diversas técnicas.

Dentro de la literatura referente a la solución del problema de SCS se ha encontrado que existen diferentes técnicas que lo resuelven. Algunas de ellas son los algoritmos de Programación Dinámica[9], aunque solo son viables cuando el lenguaje es muy pequeño y con alfabetos pequeños; los algoritmos genéticos[3], en el cual el problema de SCS tiene algunas características que lo hacen difícil de aplicar a ésta heurística, ya que la mayoría de las cadenas de longitud razonable son inválidas, la representación como una cadena nos guiará a genotipos de longitud variada y al cambiar un poco una buena solución nos dará una cadena inválida.

2.4. Heurística ACO para resolver el problema de la SCS.

La razón por la que se aplica una heurística ACO para resolver el problema de la SCS es que se trata de un problema de optimización combinatorio que se encuentra dentro de la clasificación NP-duro, por lo que es impráctico encontrar una solución óptima en un tiempo razonable. En este tipo de problemas, la manera más práctica de resolverlo es mediante el empleo de un algoritmo heurístico que produzca de manera rápida buenas soluciones, aunque éstas no sean necesariamente las óptimas, pero si muy cercanas a ésta, ya que si se emplea un método exacto, se necesita mucho tiempo para poder obtener la solución óptima, si se tiene un número de variables muy pequeño, con pocas cadenas y de longitud corta, no puede aplicarse a la solución de problemas reales, donde el número de variables es mucho mayor.

Capítulo 3

Programación Dinámica

Además de la metaheurística ACO, se pretende realizar una aplicación basada en la técnica de Programación Dinámica, que aunque ya se ha dicho que es ineficiente para encontrar la solución a este problema, nos servirá para hacer la comparación de los resultados obtenidos con la metaheurística y ver que tan cercana se encuentra de la solución óptima, para problemas con un número reducido de variables.

Existe una serie de problemas cuyas soluciones pueden ser expresadas mediante recurrencia en términos matemáticos, y posiblemente la manera más natural de resolverlos es mediante un algoritmo recursivo.

La Programación Dinámica al igual que el método divide y vencerás, resuelve problemas mediante la combinación de soluciones de subproblemas. La Programación Dinámica se aplica cuando los subproblemas no son independientes, es decir, cuando los subproblemas comparten subsubproblemas. El método divide y vencerás, parte el problema en subproblemas independientes, soluciona éstos subproblemas de manera recursiva y combina sus soluciones para resolver el problema original.

El nombre de Programación Dinámica y su metodología, lo ha tomado la teoría de algoritmos de la teoría de control, donde la Programación Dinámica es una técnica para obtener la política óptima en problemas de control con n etapas, caracterizando ésta en términos de la política óptima para un problema similar, pero con $n-1$ etapas.

El problema más importante en la teoría de control es el del control óptimo. Un proceso tiene en este contexto cuatro tipos de variables:

1. Variables independientes, que son las variables manipulables para el control del proceso.

2. Variables dependientes, que sirven para medir y describir el estado del proceso en cualquier instante de tiempo.

3. Variables producto, que se usan para indicar y medir la calidad de la tarea que realiza el sistema de control.

4. Ruidos, que son variables incontrolables que dependen del ambiente en el que el sistema lleve a cabo su operación.

El problema general del control óptimo de un proceso consiste en determinar la forma en que las variables producto pueden mantenerse en valores óptimos, independientemente de las fluctuaciones que los ruidos, o los cambios de parámetros, puedan causar. Las variables producto se usan para describir el índice de eficacia del control del sistema. Así, el problema del control óptimo supone la optimización (maximización o minimización) de ese índice de eficacia[10].

La Programación Dinámica, empieza con una pequeña porción del problema original y encuentra la solución óptima para este problema pequeño, resuelve cada subproblema solo una vez y guarda el resultado en una tabla, evitando así el trabajo de volver a calcular la solución cada vez que el subproblema es encontrado[11].

La solución de problemas mediante ésta técnica se basa en el llamado principio de óptimo enunciado por Richard E. Bellman en 1957, y que dice:

"En una secuencia de decisiones óptima, toda subsecuencia ha de ser también óptima"[10].

Para que un problema pueda ser resuelto por medio de ésta técnica, se han de cumplir dos condiciones:

- La solución al problema ha de ser alcanzada a través de una secuencia de decisiones, una en cada etapa.
- Dicha secuencia de decisiones ha de cumplir el principio de óptimo.

El desarrollo de un algoritmo de Programación Dinámica puede ser descrito mediante una secuencia de cuatro pasos:

1. Caracterizar la estructura de una solución óptima.
2. Definir recursivamente el valor de una solución óptima.

3. Calcular el valor de una solución en forma bottom-up.

4. Construir una solución óptima de la información calculada.

Los pasos del uno al tres forman la base de una solución mediante Programación Dinámica a un problema. El paso cuatro puede ser omitido sólo si el valor de una solución óptima es requerido.

La estrategia bottom-up, consiste en resolver primero los subproblemas más pequeños, almacenar su solución y luego resolver los subproblemas más complejos, usando los resultados almacenados[12].

Existen dos condiciones que se deben de cumplir antes de comenzar a buscar una solución a un problema de optimización usando la técnica de Programación Dinámica:

1. Subestructura óptima. Un problema tiene subestructura óptima cuando la solución óptima a un problema se puede componer a partir de soluciones óptimas de sus subproblemas.

2. Superposición de problemas. El cálculo de la solución óptima implica resolver muchas veces un mismo subproblema[4].

Otra forma de resolver problemas en los que no es posible conseguir una sucesión de decisiones que, etapa por etapa, formen una sucesión óptima, es intentarlo sobre todas las posibles sucesiones de decisiones, mediante el uso de la fuerza bruta. Lo que se hace es enumerar todas esas sucesiones, y entonces tomar la mejor respecto al criterio que se este usando como objetivo. La Programación Dinámica haciendo uso del principio de optimalidad de Bellman, a menudo, reduce drásticamente la cantidad de enumeraciones que hay que hacer, evitando la enumeración de algunas sucesiones que posiblemente nunca podrán ser óptimas[10].

3.1. Aplicaciones de la Programación Dinámica.

En esta sección, se presentan ejemplos de problemas resueltos mediante la aplicación de Algoritmos de Programación Dinámica, así como la justificación de su uso[12].

3.1.1. Cálculo de los Números de Fibonacci.

Se trata del cálculo de los términos de la sucesión de números de Fibonacci. Dicha sucesión podemos expresarla recursivamente en términos matemáticos de la siguiente manera:

$$Fib(n) = \begin{cases} 1,\ n = 0, 1 \\ fib(n-1) + fib(n-2), n > 1. \end{cases} \tag{3.1}$$

Por tanto, la forma más natural de calcular los términos de esa sucesión es mediante un algoritmo recursivo.

La ineficiencia del algoritmo bajo la técnica divide y vencerás, se debe a que se producen llamadas recursivas repetidas para calcular valores de la sucesión, que a pesar de que se han calculado previamente, no se conserva el resultado y por tanto es necesario volver a calcularlos cada vez que se encuentran.

Para este problema, es posible diseñar un algoritmo que lo resuelve en tiempo lineal, mediante la construcción de una tabla que permita ir almacenando los cálculos realizados hasta el momento para poder reutilizarlos posteriormente.

El uso de vectores o tablas para eliminar la repetición de los cálculos, es el punto clave de los algoritmos de Programación Dinámica.

3.1.2. El Algoritmo de Floyd.

Sea G un grafo dirigido y ponderado. Para calcular el camino mínimo entre dos vértices cualesquiera del grafo, podemos aplicar el algoritmo de Floyd que, dada la matriz L de adyacencia del grafo G, calcula una matriz D con la longitud del camino mínimo que une cada par de vértices.

Este algoritmo puede ser considerado de Programación Dinámica ya que es aplicable el principio de óptimo, que puede enunciarse para este problema de la siguiente forma: si en el camino mínimo de v_i a v_j, v_k es un vértice intermedio, los caminos de v_i a v_k y de v_k a v_j han de ser a su vez caminos mínimos. Por lo tanto, puede plantearse la relación de recurrencia que resuelve el problema como:

$$D_k(i,j) = Min_{k-1}\{D_{k-1}(i,k), D_{k-1}(k,j)\}. \tag{3.2}$$

Tal ecuación queda resuelta mediante un algoritmo que, siguiendo el esquema de Programación Dinámica, utiliza una matriz para evitar la repetición de los cálculos.

3.1.3. La Subsecuencia Común Máxima más Corta (SMCS).

La definición del problema es la siguiente: Dada una cadena α y una subsecuencia γ de α, se define sp(α, γ) como la longitud del prefijo más corto de α que es una supersecuencia de γ.

Dadas las cadenas α y β de longitudes m y n respectivamente, se define el conjunto S_{ij}, para cada $i = 1, ..., m$ y $j = 1, ..., n$, mediante:

$S_{ij} = \{(r, (x, y)) : \alpha^i$ y β^j tienen una subsecuencia común máxima γ de longitud r, y $sp(\alpha, \gamma) = x$, $sp(\alpha, \gamma) = y\}$ con $S_{00} = \{(0, (0, 0))\}$.

Para la cadena α de longitud m, posición i y símbolo α, definimos:

$$siguiente(i, a) = \begin{cases} \text{mín}\{k : \alpha[k]\} = a, k > i & \text{si existe } k \\ m + 1, & \text{en otro caso.} \end{cases} \tag{3.3}$$

Si α es una cadena, y a un símbolo del alfabeto, se denota por $\alpha + a$ a la cadena obtenida de agregar a a α. De igual manera, si el último caracter de α es a, se denota por $\alpha - a$, a la cadena obtenida de borrar del final a de α.

El algoritmo para la SMCS está basado en un esquema de Programación Dinámica para los conjuntos S_{ij} definidos. Así, la evaluación de S_{mn} revela la longitud de la SMCS. El uso de adecuados retrocesos en el arreglo de los valores S_{ij} se usa para generar la SCMS.

Capítulo 4

Algoritmos para resolver SCS

En este capítulo se describen los algoritmos utilizados para resolver el problema de la Supersecuencia Común más Corta, mediante Programación Dinámica y con la heurística ACO.

4.1. Algoritmo de Programación Dinámica.

4.1.1. Calculando la longitud de SCS.

Dadas las cadenas α y β de longitudes m y n respectivamente, definimos el conjunto T_{ij}, para cada $i = 0, ..., m$, y $j = 0, ..., n$, mediante:

$T_{ij} = \{(r, (x, y)) :$ tal que existe una supersecuencia común más corta γ de α^i y β^j, de longitud r, tal que $sp(\alpha, \gamma) = x$, $sp(\beta, \gamma) = y\}$, donde $sp(\alpha, \gamma)$ es la longitud del prefijo más corto de α que es una supersecuencia de γ, y $sp(\beta, \gamma)$ es la longitud del prefijo más corto de β que es una supersecuencia de γ.

Finalmente, para la cadena α, posición i y símbolo a, definimos:

$$f_\alpha(i, a) = \begin{cases} i+1, & \text{si } \alpha[i+1] = a \\ i, & \text{en otro caso.} \end{cases} \tag{4.1}$$

El algoritmo para SCS está basado en un esquema de Programación Dinámica para los conjuntos T_{ij} definidos anteriormente. La evaluación de T_{mn}, indica la longitud de una SCS. Además, el uso de adecuados retrocesos en el arreglo de los valores de T_{ij}, puede ser utilizado

para generar la SCS.

ALGORITMO

Dadas las cadenas α y β de longitudes m y n respectivamente, como entrada, para cada $i = 1, ..., m$ y $j = 1, ..., n$, se define la matriz T_{ij} como sigue:

(i) *Si* $\alpha[i] = \beta[j]$ *entonces*

$$T_{i,j} \longleftarrow T_{i-1,j-1} + 1.$$

(ii) *Si* $\alpha[i] \neq \beta[j]$ *entonces*

(a) *Si* $T_{i,j-1} \neq T_{i-1,j}$ *entonces*

$$T_{i,j} \longleftarrow M\acute{a}x.\{T_{i,j-1}, T_{i-1,j}\}.$$

(b) *Si* $T_{i,j-1} = T_{i-1,j}$ *entonces*

$$T_{i,j} \longleftarrow T_{i,j-1} + 1.$$

Obteniendo como salida, una matriz con las longitudes de las SCS de las cadenas α y β.

4.1.2. Recuperando la SCS.

La recuperación de la SCS requiere de un retroceso a través de la tabla de Programación Dinámica de la celda (m, n) a la celda $(0, 0)$, durante el cual la supersecuencia es construída en orden inverso. Para facilitar el retroceso, cada valor en la posición (i, j) de la tabla, debe tener asociado, durante la ejecución del algoritmo de Programación Dinámica, un apuntador indicando qué elemento en particular de las celdas $(i-1, j)$, $(i, j-1)$ o $(i-1, j-1)$ se va a incluir a la SCS.

Con esos apuntadores, cualquier ruta de un símbolo $(r, (x, y))$ en la celda (m, n) al símbolo en la celda $(0, 0)$, representa una supersecuencia común más corta de α y β de longitud r.

ALGORITMO

Como entrada, se tiene a la matriz compuesta por α, β de longitudes m y n respectivamente, para cada $i = 1, ..., m$ y $j = 1, ..., n$, con las longitudes de las cadenas α, β.

(i) *Si* $\alpha[i] = \beta[j]$ *entonces*

$T_{i,j}$ = moverse en diagonal a la izquierda *y anotar* $\alpha[i]$.

(ii) *Si* $\alpha[i] \neq \beta[j]$ *entonces*

(a) *Si* $T_{i-1,j} \geq T_{i,j-1}$ *entonces*

$T_{i,j}$ = moverse a la izquierda *y anotar* $\beta[i]$.

(b) *Si* $T_{i-1,j} < T_{i,j-1}$ *entonces*

$T_{i,j} =$ moverse hacia arriba *y anotar* $\alpha[i]$.

Como salida, se obtiene la SCS de menor longitud para las cadenas α y β, en orden inverso.

4.2. Algoritmo ACO.

Los algoritmos ACO, pueden ser aplicados a problemas de optimización discreta que tienen las siguientes características[13]:

- Un conjunto finito de N_C componentes $C = (c_1, c_2, ..., c_{N_C})$.
- Un conjunto finito L de posibles conexiones/transiciones entre los elementos de C, definido sobre un subconjunto $\hat{C}$ del producto cartesiano CxC, donde:

$$L = \{l_{c_i c_j} \mid (c_i, c_j) \in \hat{C}\}, \mid L \mid \leq Nc^2.$$

- Para cada $l_{c_i c_j} \in L$, una función de costo de conexión, bajo alguna medida de tiempo t puede ser definida.
- Un conjunto finito de restricciones $\Omega \equiv \Omega(C, L, t)$ es asignado sobre los elementos de los conjuntos C y L.
- Los estados del problema son definidos en términos de secuencias s = {c_i,c_j,...,c_k} sobre los elementos de C. S es el conjunto de todas las posibles secuencias, y $\hat{S}$ es el conjunto de todas las subsecuencias que son factibles con respecto a Ω, esto es, $\hat{S}$ es un subconjunto de S.
- La estructura de vecindad se define como sigue: el estado s_2 se dice que es un vecino de s_1 si:

  (*i*) Ambos, s_1 y s_2 están en S.

  (*ii*) Si el estado s_2 puede ser alcanzado de s_1 en un solo paso.

- Una solución ψ es un elemento de $\hat{S}$ si satisface todos los requerimientos del problema.

- Un costo $J_\psi(L,t)$ está asociado con cada solución ψ, y $J_\psi(L,t)$ es una función de todos los costos $J_{c_i c_j}$de todas las conexiones que pertenecen a la solución.

El comportamiento de las hormigas durante la ejecución del algoritmo ACO puede ser resumido como sigue: una colonia de hormigas concurrentes y asíncronas se mueven a través de estados adyacentes del problema. Las hormigas se mueven aplicando una política local de decisión que hace uso de la información contenida en las tablas de ruteo. Al moverse, las hormigas construyen soluciones al problema. Una vez que una hormiga construye una solución, o mientras la solución está siendo construída, la hormiga evalúa la solución parcial y deposita información acerca de la calidad de ésta al segregar feromona en las conexiones que ha usado. Esta información del rastro de feromona dirigirá el proceso de búsqueda de futuras hormigas. Se empleó el mismo procedimiento ACO_meta-heurística y generación y actividad de hormigas de la sección 2.2.6.

En el caso de la heurística ACO, se tienen diferentes maneras en las que se elige el símbolo que formará parte de la SCS, de acuerdo a las siguientes funciones de probabilidad:

Para Majority Merge, se emplea la siguiente:

$$p(a, v^k) \longleftarrow \frac{[\tau^k(a)]^\alpha}{\sum_{a' \in C^k} [\tau^k(a')]^\alpha}. \tag{4.2}$$

En L-Majority Merge y L-Majority Merge 2, se modifica la función que agrega la cantidad de feromona, pero se mantiene la función de probabilidad anterior:

$$\tau^k(a) \longleftarrow \sum_{i \in I^k : s_{iv_i^k} = a} \tau_{iv_i^k}(|S_i| - v_i^k + 1). \tag{4.3}$$

La diferencia entre las dos anteriores, es que en la primera, en caso de haber más de un símbolo con la misma frecuencia para ser agregados a la supersecuencia, lo elige de manera aleatoria, mientras que en la segunda, se toma en consideración la frecuencia del siguiente símbolo.

Y para Lookahead Function, la manera en la que se calcula la cantidad de feromona en las ocurrencias de un símbolo en el frente, se hace de acuerdo a la siguiente ecuación:

$$\eta(v^k, a) \longleftarrow máx_{a' \in C^k} \tau^k(a'). \tag{4.4}$$

Donde C^k y τ^k denotan el conjunto candidato y la suma de feromona en el nuevo estado que es obtenido del vector v^k si el símbolo a es elegido. Para hacer uso de este valor heurístico η, la función de probabilidad se cambia por:

$$p(a, v^k) \longleftarrow \frac{[\tau^k(a)]^\alpha \cdot [\eta(v, a)]^\beta}{\sum_{a' \in C^k} [\tau^k(a')]^\alpha \cdot [\eta(v, a')]^\beta}. \tag{4.5}$$

En la Figura 4-1 se muestra el algoritmo ACO en diagrama de flujo:

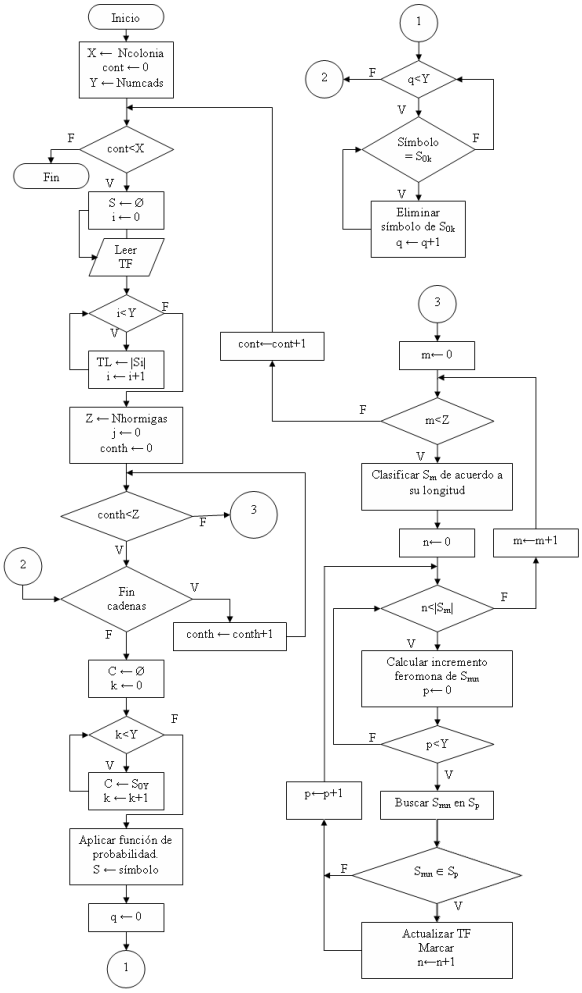

Figura 4-1: Diagrama de flujo del algoritmo ACO utilizado.

Capítulo 5

Pruebas y resultados

En este capítulo se exponen las pruebas realizadas a las aplicaciones creadas para resolver el problema de la Supesecuencia Común más Corta, así como los resultados obtenidos en cada una de ellas.

Las pruebas, se llevaron a cabo sobre un equipo con las siguientes características: procesador Intel Pentium III a 800 MHz., con 192 MB de memoria RAM. En cuanto al software, se realizaron sobre el sistema operativo Linux Mandrake 8.0 y el entorno de programación Kylix en su versión 3 Open Edition de Borland.

Inicialmente se crearon ejemplos sencillos, que se resolvieron de manera manual, en los cuales se tenían lenguajes que consistían de un número limitado de cadenas con pocos símbolos en el alfabeto (ver Apéndice B), a fin de poder comparar estos resultados con los obtenidos a través de la técnica de Programación Dinámica y el algoritmo ACO, y poder clasificarlos como buenos o malos dependiendo de lo cercano en sus resultados.

Posteriormente, se creó una aplicación para generar cadenas de manera aleatoria, cuya longitud fuera mucho mayor en comparación con las creadas al inicio de forma manual, así como con más símbolos en el alfabeto.

En las pruebas que se hicieron sobre los siguientes experimentos, fueron utilizadas diferentes técnicas, como Programación Dinámica, donde se calculó la SCS sobre los lenguajes ordenados (P.D.Ord.) y desordenados (P.D.Des.) a fin de comprobar si realmente se encuentra una SCS de menor longitud en el caso de que el lenguaje esté ordenado. Además, se emplearon cuatro metaheurísticas para compararlas con la SCS obtenida mediante Programación Dinámica, y

éstas son: Lookahead Function (L.F.), Majority Merge (M.M.), L-Majority Merge (L-M.M.) y L-Majority Merge 2 (L-M.M.2), todas ellas utilizando un algoritmo ACO.

Para cada experimento se realizaron dos tipos de prueba, una con una colonia de 15 hormigas, y otra en la que únicamente se calculaba con una hormiga.

El número de 15 hormigas para una colonia, se determinó mediante pruebas realizadas sobre colonias con diferente cantidad de hormigas: cinco, quince, treinta y cincuenta, obteniéndose que para quince hormigas se obtenían SCS de menor longitud que con cinco hormigas, y para el caso de 30 y 50, no había una disminución en las longitudes encontradas por éstas últimas, y el tiempo se incrementaba.

Los lenguajes se crearon mediante la opción *Crear Lenguaje* del programa, por lo que los símbolos que componen las cadenas son tomados de forma aleatoria, sobre los símbolos del alfabeto.

Las pruebas que se hicieron una vez terminada la aplicación, son las siguientes.

5.1. PRIMER EXPERIMENTO.

Este experimento, bajo el alfabeto $\Sigma = \{a, b\}$, consta de seis lenguajes, que son los siguientes:

L1: 10 cadenas de 8 caracteres cada una.

L2: 10 cadenas de 4 caracteres cada una.

L3: 10 cadenas de 12 caracteres cada una.

L4: 10 cadenas de 4 a 22 caracteres, incrementando de dos en dos.

L5: 10 cadenas de 20 caracteres cada una.

L6: 10 cadenas de 50 caracteres cada una.

5.1.1. Una colonia de quince hormigas

La siguiente figura y tabla, muestran las longitudes de la SCS obtenidas por cada una de las diferentes técnicas para el caso de una colonia de 15 hormigas.

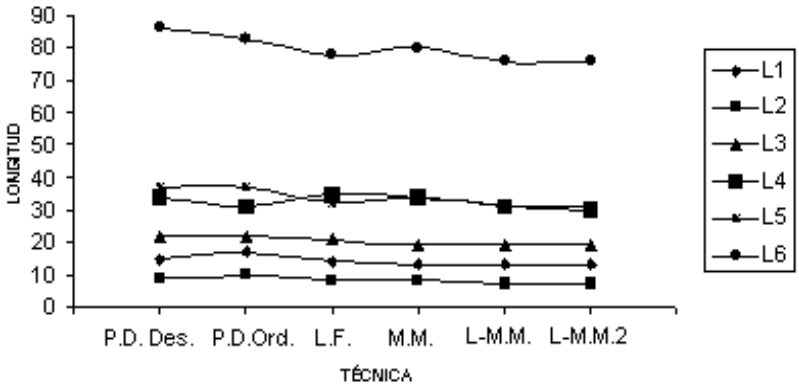

Figura 5-1: SCS encontradas para el experimento uno con quince hormigas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	15	17	14	**13**	**13**	**13**
L2	9	10	8	8	**7**	**7**
L3	22	22	21	**19**	**19**	**19**
L4	34	31	35	34	31	**30**
L5	37	37	32	34	**31**	**31**
L6	86	83	78	80	**76**	**76**

Tabla 1. Longitudes de las SCS para el primer experimento.

Los tiempos empleados en cada técnica para cada lenguaje, se muestran en la gráfica 5-2 y en la Tabla 2.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	1	3	2	**0**	1
L2	1	1	1	1	**0**	**0**
L3	**1**	2	5	**1**	**1**	**1**
L4	**2**	**2**	7	3	**2**	**2**
L5	3	3	7	4	**2**	**2**
L6	28	28	20	14	**9**	10

Tabla 2. Tiempos en milisegundos para el primer experimento.

De acuerdo a las tablas anteriores, se tiene que la que mejor desempeño obtuvo, fué la metaheurística L-Majority Merge 2, variando por muy poco L-Majority Merge. Además de ser

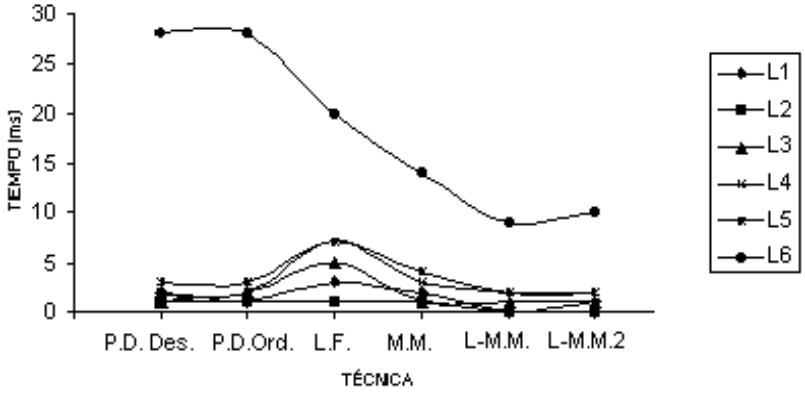

Figura 5-2: Tiempos empleados por cada técnica para el experimento uno.

las que menor tiempo emplearon para obtener las SCS.

5.1.2. Una sola hormiga

Para el caso de una sola hormiga, los resultados obtenidos por cada una de las técnicas son los siguientes:

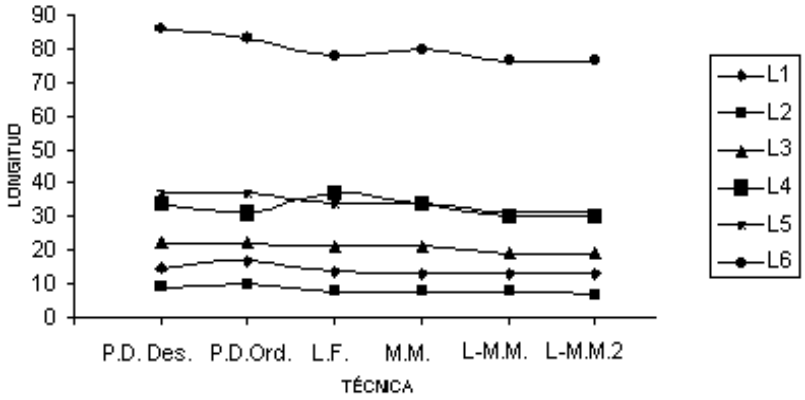

Figura 5-3: SCS encontradas por una sola hormiga para el experimento uno.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	15	17	14	**13**	**13**	**13**
L2	9	10	8	8	8	**7**
L3	22	22	21	21	**19**	**19**
L4	34	31	37	34	**30**	**30**
L5	37	37	34	34	**31**	**31**
L6	86	83	78	80	**76**	**76**

Tabla 3. Longitudes de las SCS para el primer experimento con una sola hormiga.

Los tiempos obtenidos para el caso de una sola hormiga son:

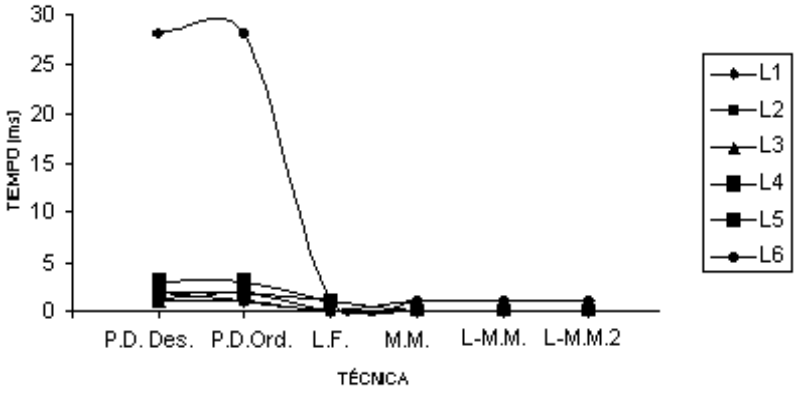

Figura 5-4: Tiempos empleados por cada técnica por una sola hormiga.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	1	**0**	1	1	1
L2	1	1	**0**	**0**	**0**	**0**
L3	1	2	**0**	**0**	**0**	**0**
L4	2	2	1	**0**	**0**	**0**
L5	3	3	1	**0**	**0**	**0**
L6	28	28	**1**	**1**	**1**	**1**

Tabla 4. Tiempos en milisegundos para el primer experimento.

Para el caso de una sola hormiga, los resultados son similares a los obtenidos para la colonia de quince hormigas: L-Majotity Merge 2 y L-Majority Merge tienen las SCS con menor longitud.

En cuanto al tiempo, los tiempos empleados por Programación Dinámica son mayores que los de las metaheurísticas.

De acuerdo a los resultados obtenidos en este experimento, donde el lenguaje consta de pocas cadenas de longitud corta, se tiene que para el caso de una colonia de quince hormigas, así como para una sola hormiga, la que mejor desempeño obtuvo, fue la metaheurística L-Majority Merge 2, variando por muy poco de L-Majority Merge. Además, las metaheurísticas obtuvieron la SCS en menor tiempo en comparación con el tomado por Programación Dinámica, donde para el caso de una sola hormiga, se nota que es mucho mayor a medida que el número de cadenas crece. El que las cadenas estén o no ordenadas, no influye en el cálculo de la SCS, pues como puede observarse, algunas veces la P.D.Des. obtuvo la más corta y otras la P.D.Ord.

Para el caso de una hormiga y una colonia de quince, la diferencia de las longitudes de las SCS no varían mucho, pero en cuanto al tiempo si, por lo que puede observarse que es mejor una sola hormiga.

5.2. SEGUNDO EXPERIMENTO.

En este experimento, se tienen tres casos, donde se varía el número de símbolos con los que cuenta el alfabeto, manteniéndose los siguientes lenguajes compuestos por diez cadenas, variando su longitud en cada uno de ellos:

L1: 10 cadenas de 4 caracteres cada una.
L2: 10 cadenas de 8 caracteres cada una.
L3: 10 cadenas de 12 caracteres cada una.
L4: 10 cadenas de 20 caracteres.

5.2.1. Primer caso.

Se tiene el siguiente alfabeto: $\Sigma = \{a, b, c, d, e\}$, y se obtuvieron los siguientes resultados, para cada uno de los siguientes subcasos.

Una colonia de quince hormigas.

La siguiente figura y tabla, muestran las longitudes de la SCS obtenidas por cada una de las diferentes técnicas para una colonia de 15 hormigas.

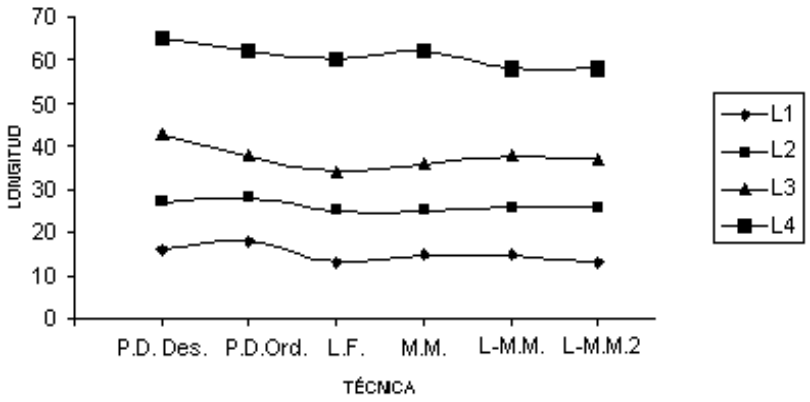

Figura 5-5: Longitudes de las SCS para el primer caso del experimento dos con quince hormigas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	16	18	**13**	15	15	**13**
L2	27	28	**25**	**25**	26	26
L3	43	38	**34**	36	38	37
L4	65	62	60	62	**58**	**58**

Tabla 5. Longitudes de las SCS encontradas por la colonia de 15 hormigas.

Los tiempos empleados para el primer caso, son los siguientes:

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	1	**0**	13	2	2	1
L2	**1**	**1**	23	3	4	2
L3	3	**2**	37	7	4	**2**
L4	6	**5**	82	10	5	**5**

Tabla 6. Tiempos en milisegundos empleados por cada técnica.

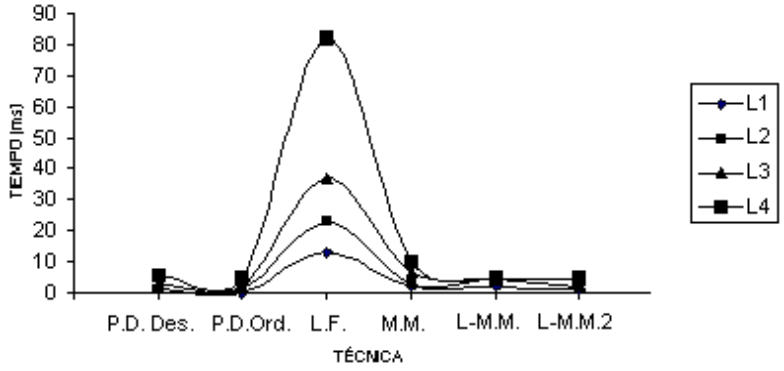

Figura 5-6: Tiempos empleados por cada técnica para el primer caso, segundo experimento.

De acuerdo a lo anterior, la metaheurística Lookahead Function, obtuvo las SCS con menor longitud en la mayoría de los lenguajes. En cuanto a los tiempos, son muy similares en las metaheurísticas, pero mucho mayores en comparación con los tomados por Programación Dinámica.

Una sola hormiga.

Para el primer caso, utilizando una sola hormiga, los resultados se muestran a continuación.

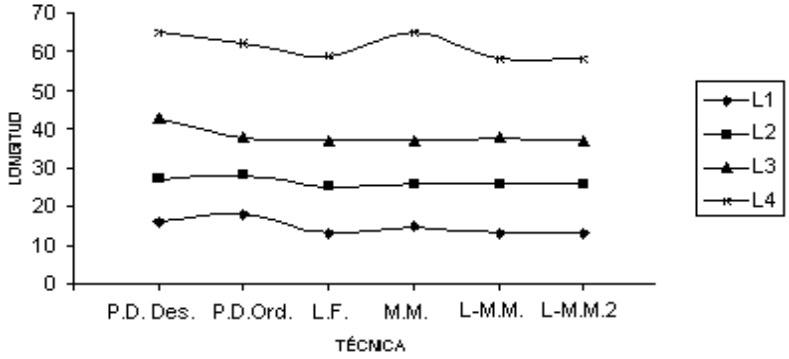

Figura 5-7: Longitudes de las SCS encontradas por una sola hormiga, para el primer caso.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	16	18	**13**	15	**13**	**13**
L2	27	28	**25**	26	26	26
L3	43	38	**37**	**37**	38	**37**
L4	65	62	59	65	**58**	**58**

Tabla 7. Longitudes de las SCS encontradas por una sola hormiga.

Los tiempos empleados para el primer caso, por una sola hormiga, son los siguientes:

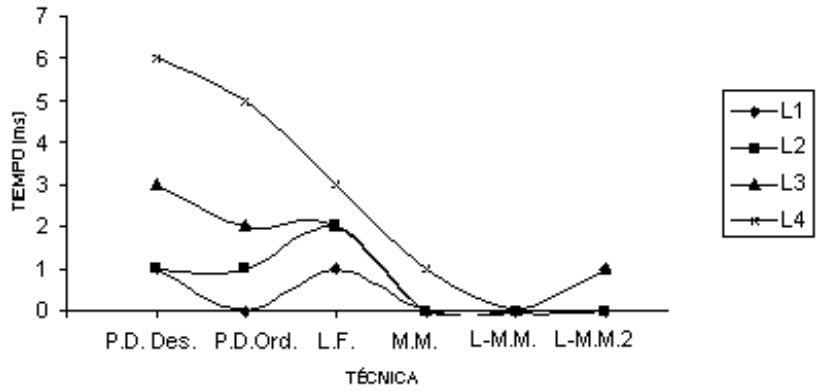

Figura 5-8: Tiempos empleados por cada técnica, para el primer caso del segundo experimento.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	1	**0**	1	**0**	**0**	1
L2	1	1	2	**0**	**0**	**0**
L3	3	2	2	**0**	**0**	1
L4	6	5	3	1	**0**	**0**

Tabla 8. Tiempos en milisegundos para el primer caso con una sola hormiga.

Para una hormiga, de las tablas anteriores, se tiene que la metaheurística Lookahead Function, encontró las SCS con menor longitud, aunque los tiempos de ésta son ligeramente menores en comparación con los empleados por Programación Dinámica, son mayores a los de otras técnicas.

Para este primer caso, se cuenta con un alfabeto un poco mayor que en el primer experimento, resultando que para la colonia de quince hormigas, la que mejor desempeño obtuvo fué

la metaheurística Lookahead Function, y para el caso de una hormiga, hubo un mismo número de SCS de menor longitud encontradas por Lookahead Function y L-Majority Merge 2. Para este caso, si el lenguaje contiene cadenas de longitud grande, es mejor que esté ordenado para obtener una SCS de menor longitud, lo que no pasa si se trata de lenguajes de longitud pequeña, para la técnica de Programación Dinámica.

Para el caso de una colonia de quince hormigas y una sola hormiga, se tiene que las longitudes de las SCS no varían mucho, pero el tiempo si se incrementa demasiado.

5.2.2. Segundo caso.

En éste segundo caso, se incrementó el número de símbolos del alfabeto:
$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m\}$, y se obtuvieron los siguientes resultados:

Una colonia de quince hormigas.

Las longitudes de la SCS obtenidas por cada una de las diferentes técnicas para el segundo caso con una colonia de 15 hormigas.

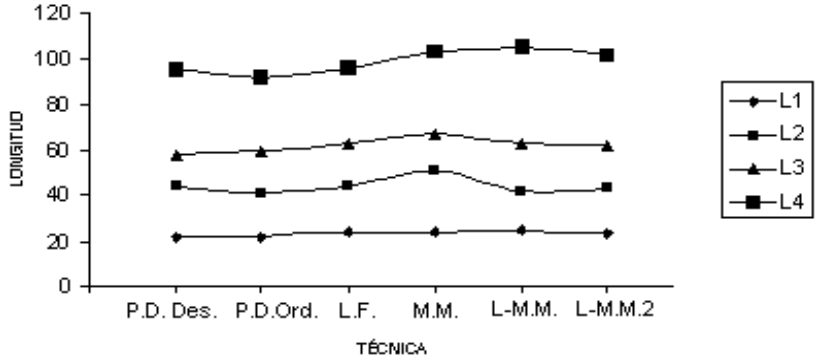

Figura 5-9: Longitudes de las SCS encontradas por cada técnica para el segundo caso.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**22**	**22**	24	24	25	23
L2	44	**41**	44	51	42	43
L3	**58**	59	63	67	63	62
L4	95	**92**	96	103	105	102

Tabla 9. Longitudes de las SCS encontradas por la colonia de 15 hormigas.

Los tiempos empleados para el segundo caso, son los siguientes:

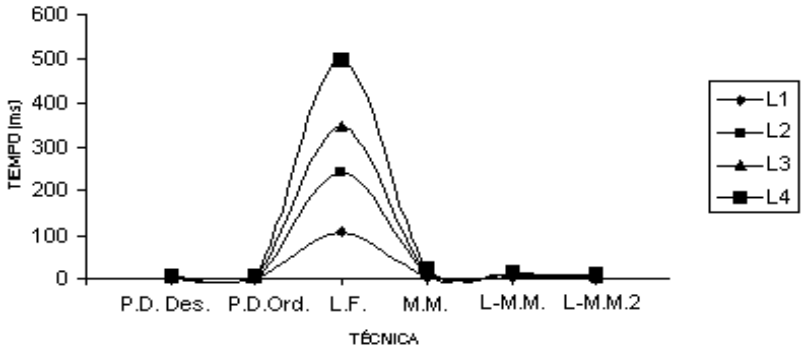

Figura 5-10: Tiempos empleados por cada técnica, para el segundo caso.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**0**	1	106	5	5	2
L2	2	**1**	242	13	6	4
L3	**2**	3	347	15	9	5
L4	8	**7**	499	24	15	9

Tabla 10. Tiempos en milisegundos para el segundo caso.

En este segundo caso, con una colonia de quince hormigas, la técnica de Programación Dinámica obtuvo las SCS de menor longitud, y sus tiempos son menores que todas las metaheurísticas.

Una sola hormiga.

Para el segundo caso, utilizando una sola hormiga, los resultados se muestran a continuación.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**22**	**22**	23	25	23	23
L2	44	**41**	45	48	43	43
L3	**58**	59	62	67	63	62
L4	95	**92**	100	107	106	102

Tabla 11. Longitudes de las SCS encontradas por una sola hormiga.

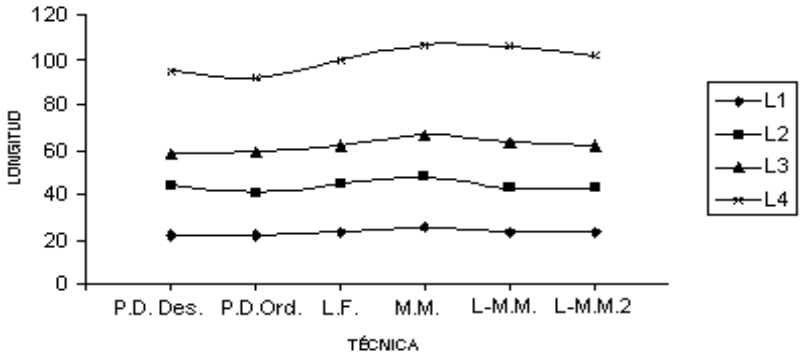

Figura 5-11: Longitudes de las SCS encontradas por una sola hormiga para el segundo caso.

Los tiempos empleados para el segundo caso, por una sola hormiga, son los siguientes:

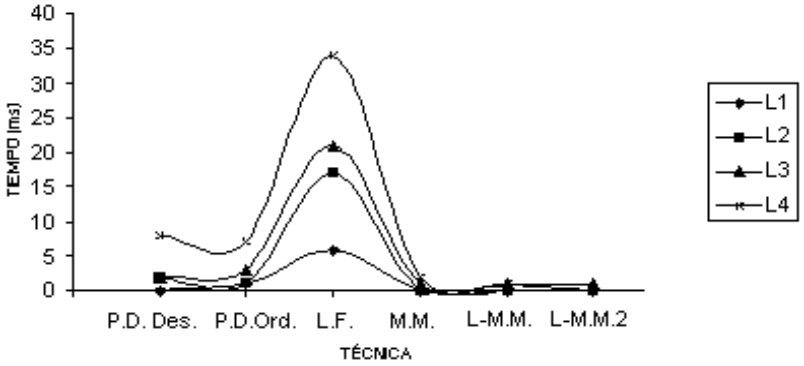

Figura 5-12: Tiempos empleados por cada técnica.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**0**	1	6	**0**	**0**	**0**
L2	2	1	17	**0**	**0**	**0**
L3	2	3	21	**1**	**1**	**1**
L4	8	7	34	2	1	**0**

Tabla 12. Tiempos en milisegundos para el segundo caso con una sola hormiga.

Aquí, se observa que la técnica de Programación Dinámica sigue obteniendo las SCS con menor longitud, aunque los tiempos empleados por la mayoría de las metaheurísticas son menores que los de Programación Dinámica, y los resultados de éstas son cercanas a las SCS menores obtenidas.

Para este caso, donde se incrementa el número de símbolos del que consta el alfabeto, se observa que para una colonia de hormigas y para una sola hormiga, la que mejor resultados arrojó, fue la técnica de Programación Dinámica, aunque éstos variaban muy poco en longitud respecto de los encontrados por las metaheurísticas. En cuanto al tiempo, la técnica de Programación Dinámica, para el caso de quince hormigas, son mucho menores a los de las metaheurísticas, y para una sola hormiga, son ligeramente mayores a éstas.

En este caso, se tiene que si el lenguaje está ordenado y tiene un número grande de cadenas, con un alfabeto grande, se encuentra una SCS de menor longitud por Programación Dinámica.

Además, las longitudes de las SCS obtenidas por una colonia de quince hormigas y una sola, varían a lo más en cuatro unidades, lo que no significa mucho, pero el tiempo si se incrementa de manera notable.

5.2.3. Tercer caso.

En éste tercer caso, se incrementó el número de símbolos del alfabeto:

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}.$$

Una colonia de quince hormigas.

Las longitudes de la SCS obtenidas por cada una de las diferentes técnicas para el tercer caso con una colonia de 15 hormigas, se muestran a continuación.

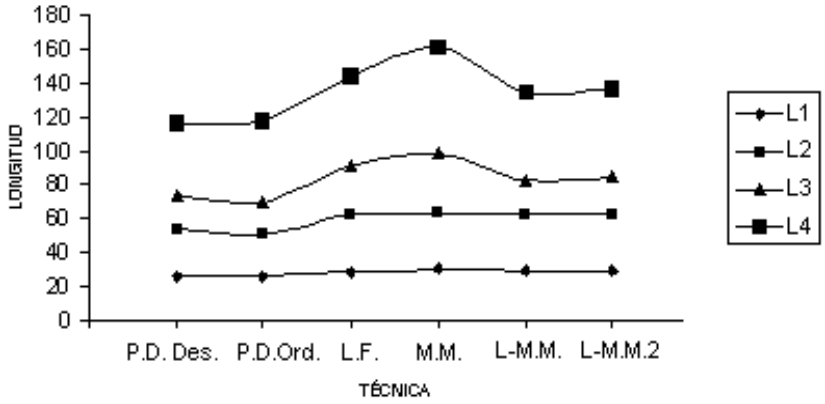

Figura 5-13: Longitudes de las SCS para el tercer caso con quince hormigas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**27**	**27**	29	31	30	30
L2	54	**51**	62	64	62	62
L3	74	**70**	91	99	83	85
L4	**116**	118	144	161	135	137

Tabla 13. Longitudes de las SCS encontradas por la colonia de 15 hormigas.

Los tiempos empleados para el tercer caso, por una colonia de quince hormigas, son los siguientes:

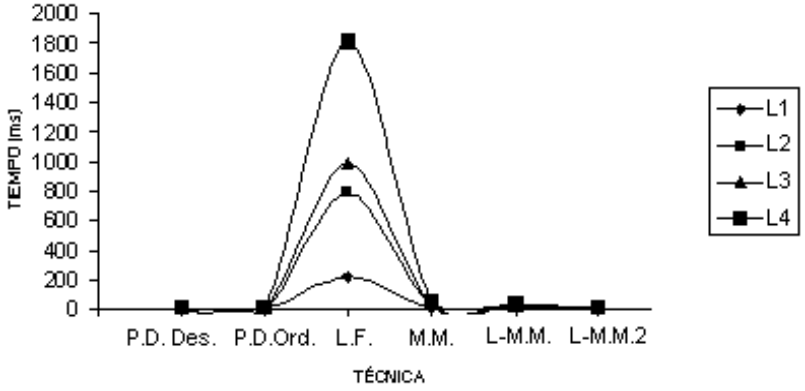

Figura 5-14: Tiempos empleados por cada técnica, para el tercer caso, con quince hormigas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**1**	**1**	221	9	8	3
L2	**1**	2	785	19	19	7
L3	**3**	**3**	983	41	20	10
L4	**8**	9	1810	64	31	16

Tabla 14. Tiempos en milisegundos para el tercer caso.

Con quince hormigas, se tiene que la que mejor SCS se obtuvo mediante Programación Dinámica, e igual que en el caso anterior, los tiempos de ésta son los menores. Además de que la longitud de las SCS obtenidas por las metaheurísticas son mucho mayores que las de Programación Dinámica.

Una sola hormiga.

Para el segundo caso, utilizando una sola hormiga, se obtuvo:

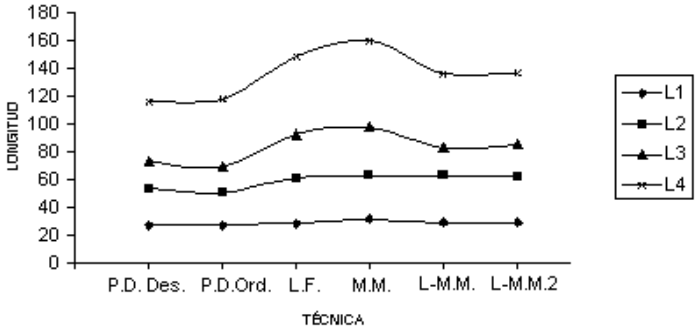

Figura 5-15: Longitudes de las SCS encontradas por una sola hormiga del tercer caso.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**27**	**27**	28	32	30	30
L2	54	**51**	61	63	63	62
L3	74	**70**	93	98	83	85
L4	**116**	118	148	160	136	137

Tabla 15. Longitudes de las SCS encontradas por una sola hormiga.

Los tiempos empleados para el tercer caso, por una sola hormiga, son los siguientes:

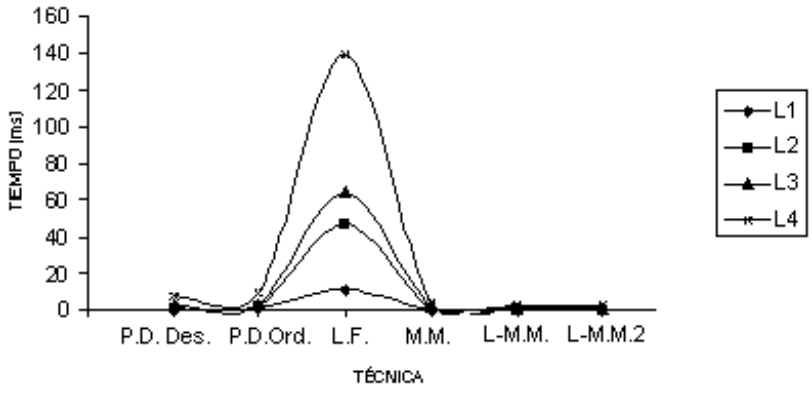

Figura 5-16: Tiempos empleados por cada técnica, para el tercer caso.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**0**	1	11	**0**	1	**0**
L2	1	2	47	1	1	**0**
L3	3	3	64	2	**1**	**1**
L4	8	9	139	4	**3**	**3**

Tabla 16. Tiempos en milisegundos para el tercer caso con una sola hormiga.

De las tablas anteriores, la que mejores SCS encontró, fue Programación Dinámica, aunque los tiempos de las metaheurísticas disminuyeron y en algunos casos, las longitudes son cercanas a las menores encontradas. La técnica de Programación Dinámica, obtiene SCS de menor longitud, si el lenguaje está ordenado, para el caso de lenguajes con longitud grande y muchos símbolos en su alfabeto.

En este tercer caso, sucede lo mismo que en los casos anteriores, que la diferencia no es mucha en las SCS obtenidas por una sola hormiga y por una colonia de quince, pero en el tiempo si hay una diferencia considerable.

En general, para este segundo experimento, se puede observar que para el caso de que se tenga un número reducido de cadenas de longitud pequeña, y con un alfabeto grande, la técnica que encuentra la SCS de menor tamaño, es Programación Dinámica. Además de que a medida que se incrementa el número de símbolos del que está formado el lenguaje, es mejor que esté ordenado, ya que así se obtiene una SCS de menor longitud.

En cuanto a la colonia de quince hormigas y una sola, no hay mucha diferencia en sus longitudes, pero en el tiempo existe una gran diferencia para el primer caso respecto del segundo.

5.3. TERCER EXPERIMENTO.

En este experimento, se tiene un alfabeto de tres símbolos $\Sigma = \{a, b, c\}$ y una longitud constante de doce símbolos en las cadenas de los lenguajes. Lo que se varió, es el número de cadenas que los componían, teniéndose lo siguiente:

L1: 10 cadenas.	L7: 150 cadenas.
L2: 30 cadenas.	L8: 200 cadenas.
L3: 50 cadenas.	L9: 300 cadenas.
L4: 70 cadenas.	L10: 500 cadenas.
L5: 90 cadenas.	L11: 700 cadenas.
L6: 100 cadenas.	L12: 1000 cadenas.

5.3.1. Una colonia de quince hormigas.

A continuación se muestran las longitudes de la SCS obtenidas para cada lenguaje, así como los tiempos que se tardó cada una de las técnicas en obtenerla.

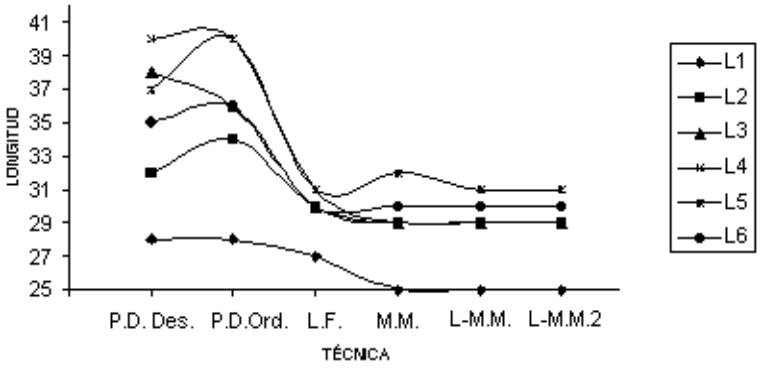

Figura 5-17: Longitudes de las SCS encontradas por cada técnica para el tercer experimento.

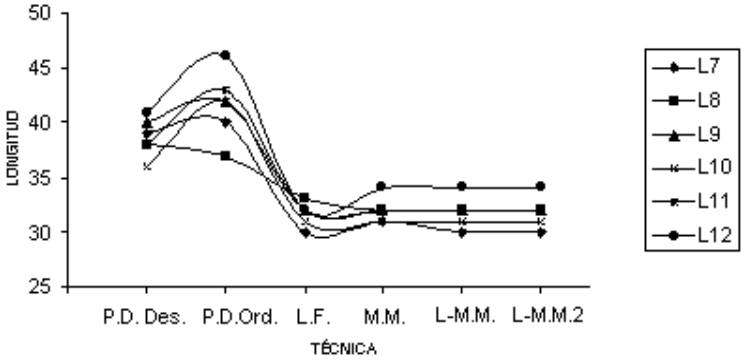

Figura 5-18: Longitudes de las SCS encontradas para el tercer experimento (continuación).

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	28	28	27	**25**	**25**	**25**
L2	32	34	30	**29**	**29**	**29**
L3	38	36	30	**29**	**29**	**29**
L4	40	40	31	**29**	**29**	**29**
L5	37	40	**31**	32	**31**	**31**
L6	35	36	**30**	**30**	**30**	**30**
L7	39	40	**30**	31	**30**	**30**
L8	38	37	33	**32**	**32**	**32**
L9	40	42	**32**	**32**	**32**	**32**
L10	36	42	**31**	**31**	**31**	**31**
L11	38	43	**32**	**32**	**32**	**32**
L12	41	46	**32**	34	34	34

Tabla 17. Longitudes de las SCS para el tercer experimento.

Los tiempos empleados para encontrar la SCS por cada una de las técnicas se muestran a continuación:

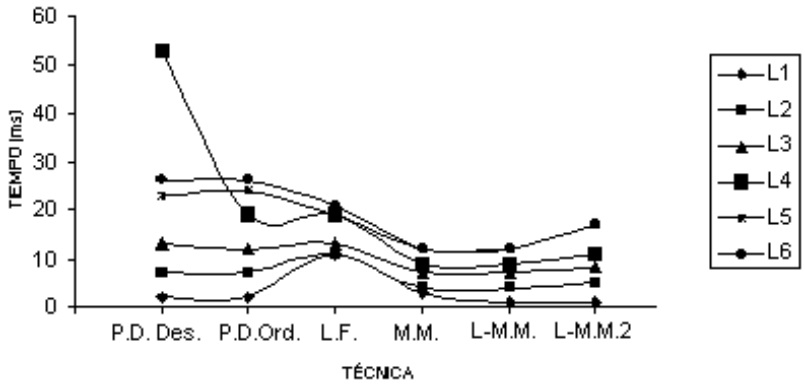

Figura 5-19: Tiempos empleados por cada técnica, con una colonia de quince hormigas.

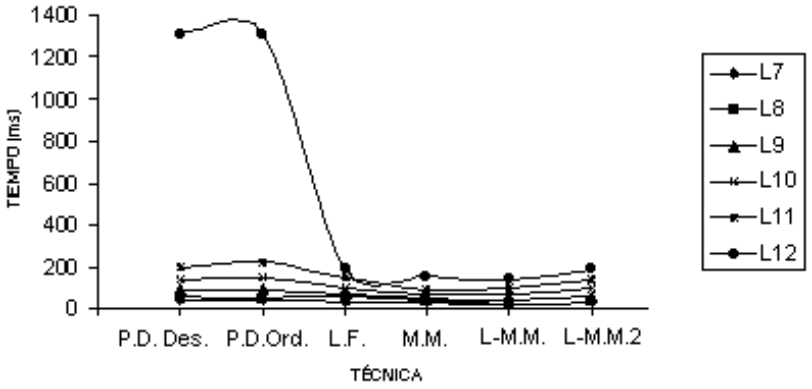

Figura 5-20: Tiempos empleados para una colonia de quince hormigas (continuación).

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	2	11	3	**1**	**1**
L2	7	7	11	**4**	**4**	5
L3	13	12	13	**7**	**7**	8
L4	53	19	19	**9**	**9**	11
L5	23	24	19	**12**	**12**	17
L6	26	26	21	**12**	**12**	17
L7	42	39	29	30	**19**	36
L8	56	53	57	37	**28**	34
L9	88	90	73	52	**38**	57
L10	138	147	100	**66**	67	95
L11	195	222	148	**92**	99	136
L12	1:306	1:311	188	154	**143**	185

Tabla 18. Tiempos en segundos y milisegundos para el tercer experimento.

Para el caso de una colonia de quince hormigas, se tiene que las metaheurísticas encontraron las SCS de menor longitud, sobresaliendo Majority Merge, L-Majority Merge y L-Majority Merge 2. Los tiempos empleados por Programación Dinámica, son mucho mayores que los tomados por las metaheurísticas.

5.3.2. Una sola hormiga.

Para el caso de una sola hormiga, las longitudes de las SCS encontradas, se muestran en las siguientes figuras y tabla.

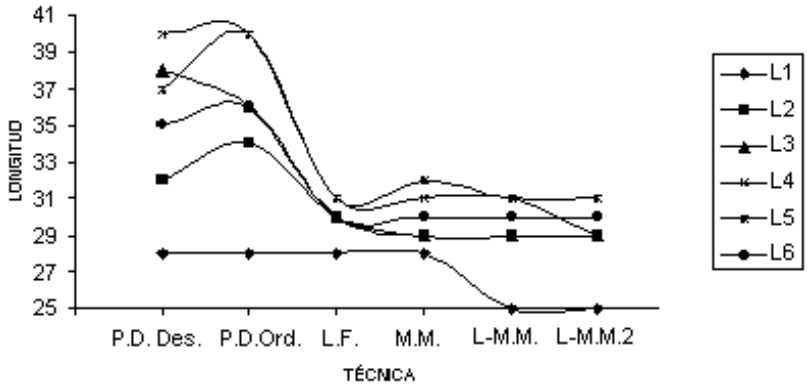

Figura 5-21: Longitudes de las SCS del tercer experimento con una sola hormiga.

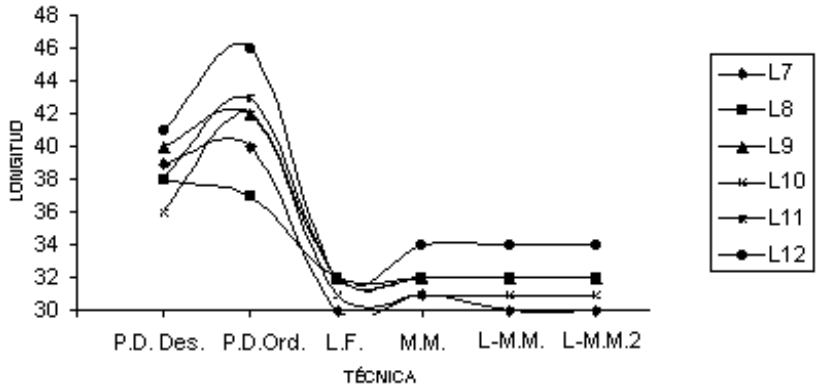

Figura 5-22: Longitudes de las SCS del tercer experimento con una sola hormiga (continuación).

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	28	28	28	28	**25**	**25**
L2	32	34	30	**29**	**29**	**29**
L3	38	36	30	**29**	**29**	**29**
L4	40	40	31	31	31	**29**
L5	37	40	**31**	32	**31**	**31**
L6	35	36	**30**	**30**	**30**	**30**
L7	39	40	**30**	31	**30**	**30**
L8	38	37	**32**	**32**	**32**	**32**
L9	40	42	**32**	**32**	**32**	**32**
L10	36	42	**31**	**31**	**31**	**31**
L11	38	43	**32**	**32**	**32**	**32**
L12	41	46	**32**	34	34	34

Tabla 19. Longitudes de las SCS para el tercer experimento con una sola hormiga.

Los tiempos empleados para cada lenguaje, se muestran a continuación:

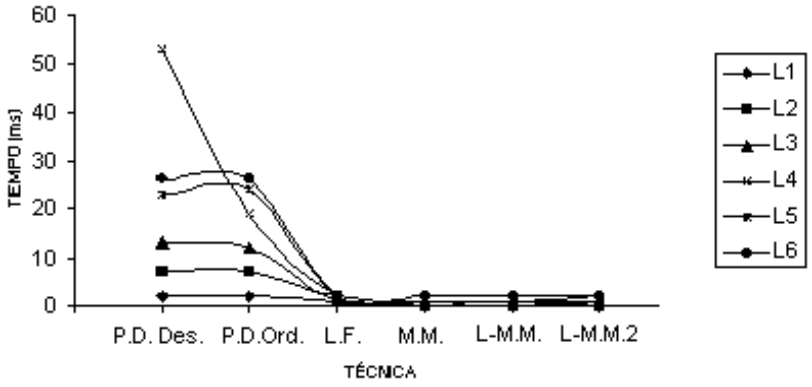

Figura 5-23: Tiempos empleados por cada técnica, para el tercer experimento.

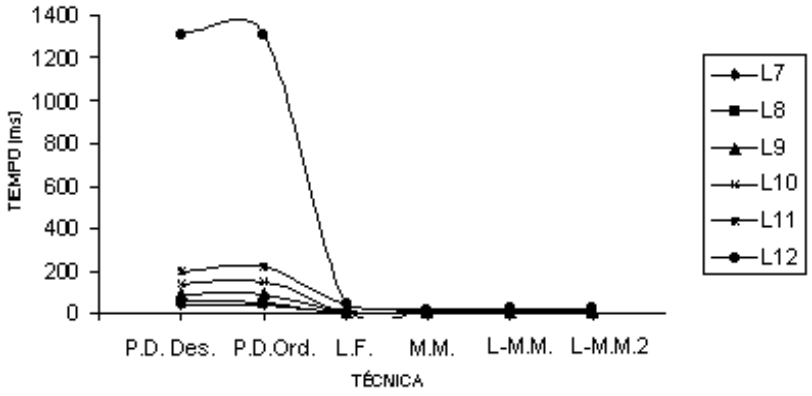

Figura 5-24: Tiempos empleados para el tercer experimento (continuación).

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	2	1	**0**	1	**0**
L2	7	7	2	**0**	**0**	**0**
L3	13	12	1	**0**	**0**	1
L4	53	19	2	**1**	**1**	**1**
L5	23	24	2	**1**	**1**	2
L6	26	26	**2**	**2**	**2**	**2**
L7	42	39	3	**2**	3	3
L8	56	53	4	**3**	**3**	4
L9	88	90	7	**6**	**6**	7
L10	138	147	12	**11**	13	12
L11	195	222	37	**13**	25	16
L12	1:306	1:311	41	**20**	21	27

Tabla 20. Tiempos en segundos y milisegundos para el tercer experimento con una sola hormiga.

Para el caso de una hormiga en éste experimento, las SCS encontradas por las metaheurísticas y en particular la L-majority Merge 2, son mucho menores en algunos casos que las de Programación Dinámica, y los tiempos de las metaheurísticas, para todos los casos son menores que los de ésta última.

En este tercer experimento, se tienen lenguajes con una misma longitud, y un alfabeto pequeño, incrementándose el número de cadenas. Para el caso de una colonia de quince hormigas y una sola hormiga, se observa que las metaheurísticas tienen un mejor desempeño variando en una o dos unidades entre ellas, en comparación con la técnica de Programación Dinámica. Y el tiempo empleado por ésta última, es mucho mayor que la de las metaheurísticas.

Para el caso de que se tengan muchas cadenas de longitud pequeña y pocos símbolos en su alfabeto, las SCS de menor longitud, obtenidas por Programación Dinámica, son de un lenguaje desordenado.

En cuanto a la diferencia de la longitud de las SCS, se tiene que es muy poca para el caso de una colonia de quince hormigas y una sola hormiga, aunque en el tiempo existe un incremento considerable.

5.4. CUARTO EXPERIMENTO.

Para este experimento, se tienen dos casos en los que se hizo una combinación de todos los factores, como el número de cadenas, su longitud y el número de símbolos que componen el alfabeto. Los lenguajes son:

L1: 10 cadenas de longitud 8.

L2: 10 cadenas de longitud 30.

L3: 100 cadenas de longitud 50.

L4: 100 cadenas de longitud 100.

L5: 500 cadenas de longitud 20.

L6: 500 cadenas de longitud 50.

L7: 1000 cadenas de longitud 150.

5.4.1. Primer caso.

En el primer caso, se tiene el alfabeto $\Sigma = \{a, b, c, d, e, f, g, h, i\}$, obteniéndose los resultados que se muestran a continuación.

Una colonia de quince hormigas.

Para este caso, las longitudes de las SCS encontradas se muestran en las siguientes figuras.

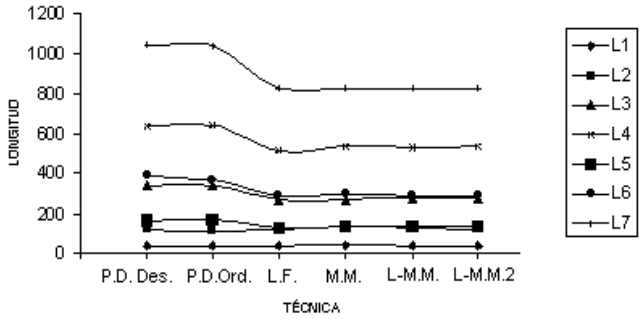

Figura 5-25: Longitudes de las SCS encontradas por cada técnica para el cuarto experimento.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	36	36	**32**	39	36	33
L2	121	**113**	121	133	126	118
L3	342	337	**269**	271	273	273
L4	632	639	**514**	534	530	534
L5	164	170	**130**	133	133	133
L6	387	365	**287**	297	292	292
L7	1046	1038	825	825	**823**	**823**

Tabla 21. Longitudes de las SCS para el cuarto experimento primer caso.

Los tiempos empleados para cada lenguaje, se muestran a continuación:

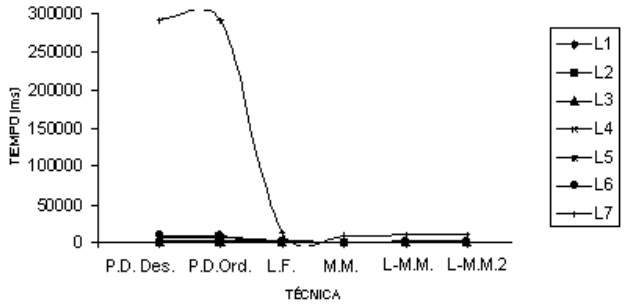

Figura 5-26: Tiempos para cada técnica para el primer caso, con quince hormigas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	**1**	107	9	5	2
L2	15	16	391	44	15	**10**
L3	1:345	1:325	663	**197**	183	279
L4	7:574	7:671	1:433	502	**488**	559
L5	1:058	1:081	593	**287**	308	390
L6	8:501	8:021	1:572	**857**	905	1:095
L7	4:51:594	4:51:834	13:162	**9:619**	9:832	10:800

Tabla 22. Tiempos en minutos, segundos y milisegundos empleados por cada técnica.

De las tablas anteriores, se observa que las SCS de menor tamaño, son las encontradas por la metaheurística Lookahead Function, y aunque los tiempos son mayores que las de las otras metaheurísticas, son mucho menores en comparación con la técnica de Programación Dinámica.

Una sola hormiga.

Las longitudes de las SCS encontradas se muestran en las siguientes figuras.

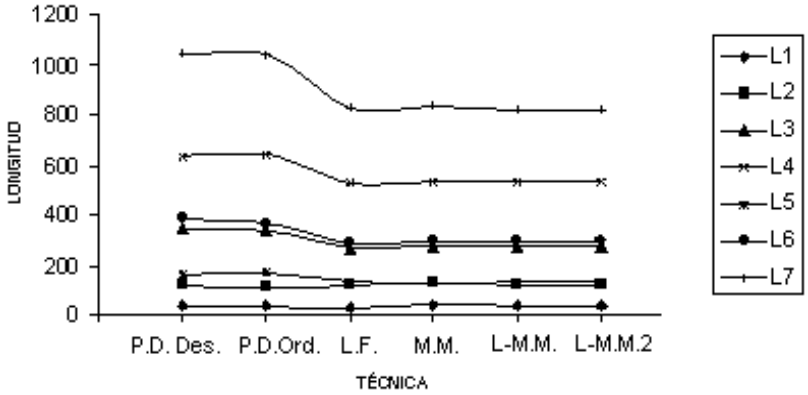

Figura 5-27: Longitudes de las SCS encontradas por cada técnica con una sola hormiga.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	36	36	**30**	41	33	33
L2	121	**113**	121	124	118	118
L3	342	337	**270**	275	273	273
L4	632	639	**526**	530	530	534
L5	164	170	131	**129**	133	133
L6	387	365	**289**	295	294	292
L7	1046	1038	825	836	**823**	**823**

Tabla 23. Longitudes de las SCS para el cuarto experimento primer caso.

Los tiempos de cada técnica se muestran a continuación:

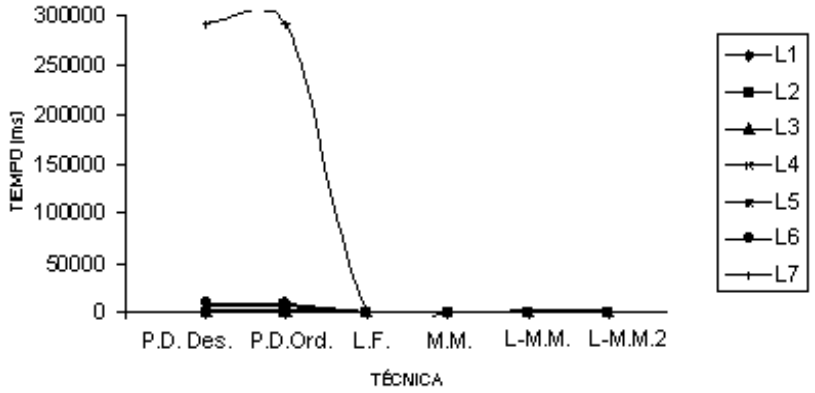

Figura 5-28: Tiempos para cada técnica para el primer caso, del cuarto experimento.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	1	4	1	**0**	**0**
L2	15	16	27	2	**1**	**1**
L3	1:345	1:325	45	**14**	15	19
L4	7:574	7:671	107	49	**43**	56
L5	1:058	1:081	43	**22**	26	34
L6	8:501	8:021	119	**73**	84	98
L7	4:51:594	4:51:834	1:076	**877**	904	1:000

Tabla 24. Tiempos en minutos, segundos y milisegundos empleados por cada técnica.

Para una sola hormiga, se tiene que la SCS de menor longitud se obtuvo por la metaheurística Lookahead Function, y los tiempos disminuyen considerablemente en comparación con los de Programación Dinámica.

Para este primer caso, se hace una combinación de número de cadenas, longitud y tamaño del alfabeto, observándose que la metaheurística que mejor desempeño tiene es Lookahead Function en la mayoría de los casos, para el caso de una colonia de quince hormigas y una sola hormiga. Y en el caso de los tiempos, éstos son menores que los empleados por Programación Dinámica. Si el lenguaje está ordenado, la técnica de Programación Dinámica encuentra una SCS de menor longitud.

Además, la diferencia en las longitudes de las SCS encontradas por la colonia de quince hormigas y una sola, es mínima, pero en el tiempo se aprecia un incremento considerable.

5.4.2. Segundo caso.

En este segundo caso, se mantienen los lenguajes como en el caso uno, excepto que el número de símbolos en el alfabeto se aumentó:

$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, obteniéndose los siguientes resultados.

Una colonia de quince hormigas.

Para este caso, las longitudes de las SCS encontradas se muestran en las siguientes figuras.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**50**	55	58	66	58	56
L2	166	**165**	222	238	217	223
L3	762	763	**646**	676	668	665
L4	1453	1459	**1240**	1316	1280	1286
L5	414	422	**332**	336	333	333
L6	963	976	**747**	772	769	764
L7	2805	2813	**2165**	2212	2184	2193

Tabla 25. Longitudes de las SCS para el cuarto experimento segundo caso.

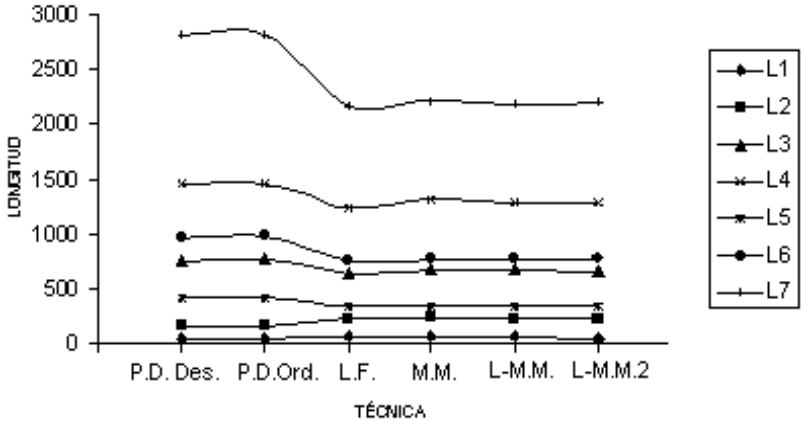

Figura 5-29: Longitudes de las SCS encontradas por cada técnica, para el segundo caso.

Los tiempos se muestran a continuación.

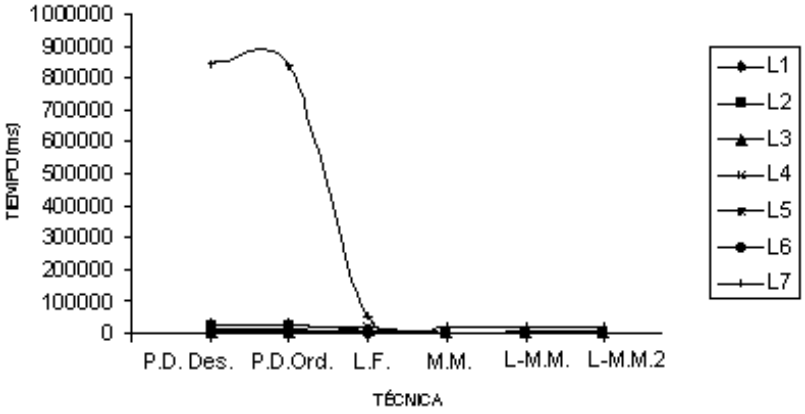

Figura 5-30: Tiempos para cada técnica para el segundo caso, con quince hormigas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**2**	**2**	603	23	15	7
L2	**19**	20	2:918	92	58	29
L3	3:101	3:080	11:052	629	**524**	629
L4	15:860	16:020	20:963	1:412	**1:154**	1:390
L5	3:522	3:547	5:390	**1:052**	1:091	1:331
L6	24:320	25:726	11:448	**2:677**	2:742	3:351
L7	14:01:924	14:00:376	51:189	20:573	**19:411**	23:180

Tabla 26. Tiempos en minutos, segundos y milisegundos empleados por cada técnica.

Para una colonia de quince hormigas, la metaheurística Lookahead Function obtuvo los mejores resultados, es decir, las SCS de menor longitud para la mayoría de los casos. En cuanto al tiempo, son mucho menores a los empleados por Programación Dinámica.

Una sola hormiga.

Las longitudes de las SCS encontradas se muestran en las siguientes figuras.

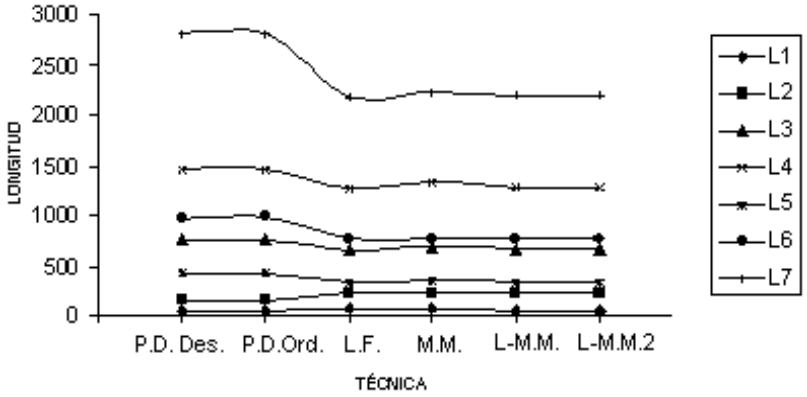

Figura 5-31: Longitudes de las SCS encontradas por cada técnica, con una sola hormiga.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**50**	55	62	66	58	56
L2	166	**165**	225	236	228	223
L3	762	763	**656**	690	667	665
L4	1453	1459	**1262**	1340	1282	1286
L5	414	422	334	346	339	**333**
L6	963	976	774	773	770	**764**
L7	2805	2813	**2173**	2224	2197	2193

Tabla 27. Longitudes de las SCS para el cuarto experimento segundo caso.

Los tiempos empleados por cada una de las técnicas, se muestran a continuación:

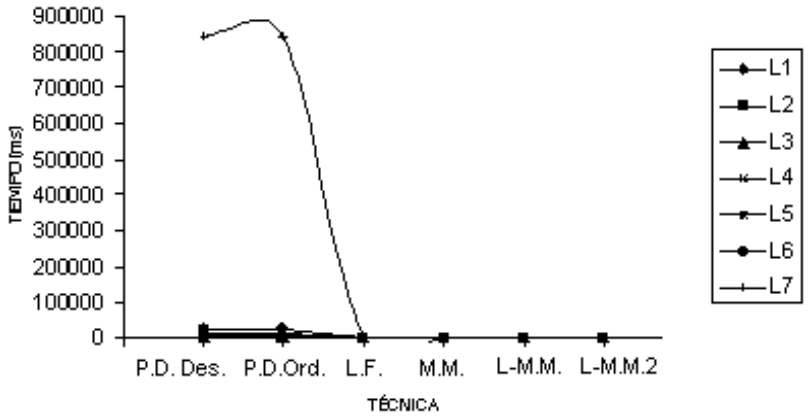

Figura 5-32: Tiempos para cada técnica para el segundo caso del cuarto experimento.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	2	2	41	2	**1**	**1**
L2	19	20	203	5	4	**2**
L3	3:101	3:080	703	**43**	48	44
L4	15:860	16:020	1:419	106	**96**	109
L5	3:522	3:547	367	**79**	84	103
L6	24:320	25:726	874	**193**	203	247
L7	14:01:924	14:00:376	3:611	**1:568**	1:638	1:881

Tabla 28. Tiempos en minutos, segundos y milisegundos empleados por cada técnica.

Aquí, se tiene que dos de las metaheurísticas obtuvieron los mejores resultados: Lookahead Function y L-Majority Merge 2, y los tiempos son mucho menores que los de Programación Dinámica. Si se incrementa demasiado el número de símbolos del que consta el alfabeto, es mejor que el lenguaje se encuentre desordenado para obtener por Programación Dinámica una SCS menor.

Para este caso, se tiene que la diferencia de la longitud de la SCS encontrada por una colonia de quince hormigas y una sola, sí es considerable, aunque el tiempo también se incrementa demasiado.

Para este experimento, donde se incrementó el número de símbolos del alfabeto, se tiene que de manera general, la que encontró las SCS de menor longitud fue la metaheurísitca Lookahead

Function, para los casos de una colonia de quince hormigas y una sola hormiga, sobre todo cuando hay un número de cadenas grande. Y los tiempos son mucho menores, para el caso de las metaheurísticas. Además de que si el alfabeto no es tan grande, es mejor que esté ordenado para que por medio de Programación Dinámica se encuentre una SCS de menor longitud. Si se incrementa considerablemente el número de símbolos del alfabeto, es mejor que el lenguaje esté desordenado.

De manera general, se tiene que la diferencia de longitudes obtenidas por una colonia de quince hormigas y una sola, es mayor a medida que se incrementa el número de símbolos con que cuenta el alfabeto, aunque el tiempo, siempre es mayor para el primer caso.

5.5. QUINTO EXPERIMENTO.

Para este experimento, sólo se hicieron pruebas utilizando Programación Dinámica con lenguajes desordenados y Lookahead Function para una hormiga, sobre los siguientes lenguajes, con el alfabeto $\Sigma = \{a, b, c, d, e, f, g, h, i, j\}$.

L1: 10,000 cadenas de longitud 200.

L2: 10,000 cadenas de longitud 300.

L3: 10,000 cadenas de longitud 400.

L4: 10,000 cadenas de longitud 500.

L5: 20,000 cadenas de longitud 200.

L6: 20,000 cadenas de longitud 300.

L7: 20,000 cadenas de longitud 400.

A continuación se muestran los resultados obtenidos para los dos casos:

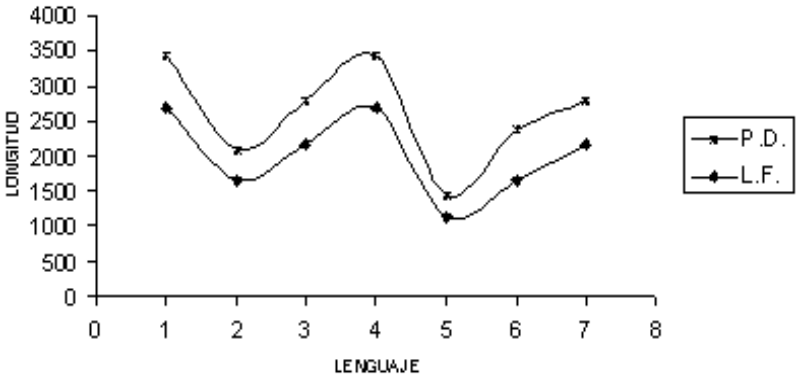

Figura 5-33: Longitudes de las SCS encontradas por cada técnica para el quinto experimento.

Lenguaje	P.D.Des.	L.F.
L1	3445	**2693**
L2	2099	**1641**
L3	2781	**2167**
L4	3445	**2693**
L5	1451	**1135**
L6	2370	**1658**
L7	2789	**2176**

Tabla 29. Longitudes de las SCS encontradas para cada lenguaje.

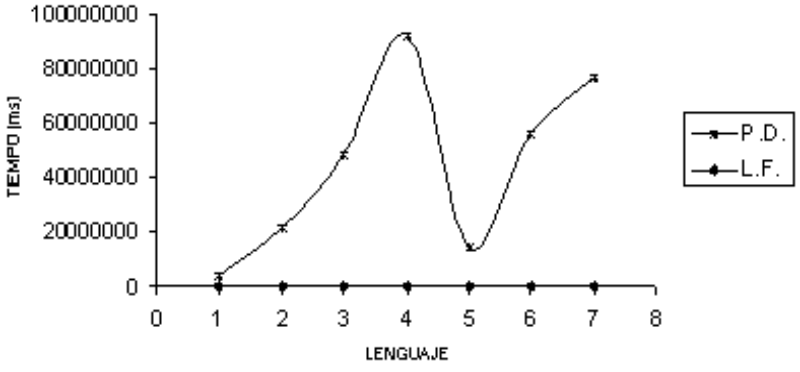

Figura 5-34: Tiempos empleados para el quinto experimento.

Lenguaje	P.D.Des.	L.F.
L1	1:58:00:897	**21:350**
L2	5:55:23:224	**45:350**
L3	13:27:15:597	**1:06:494**
L4	1:01:39:02:758	**1:37:611**
L5	3:52:10:803	**48:309**
L6	15:40:31:810	**4:49:379**
L7	21:14:54:485	**2:12:702**

Tabla 30. Tiempos en días, horas, minutos, segundos y milisegundos empleados para obtener la SCS.

Para este último experimento, se realizó únicamente para la técnica de Programación Dinámica y la metaheurística Lookahead Function para una hormiga, encontrándose que la primera, tarda mucho tiempo y las SCS que encuentra, son mucho mayores en comparación con las obtenidas por la segunda.

Capítulo 6

Conclusiones

En este capítulo, se exponen las conclusiones a las que se llegó a partir de los resultados obtenidos en las diferentes pruebas realizadas.

Para cuando se tienen lenguajes con pocas cadenas, de longitud pequeña y muy pocos símbolos en el alfabeto, las metaheurísticas encuentran las SCS de menor longitud, en particular, L-M.M.2.

Cuando se incrementa el número de símbolos del que consta el lenguaje, se observa que la metaheurística Lookahead Function, encuentra las SCS menores.

A medida que se sigue incrementando el número de símbolos del alfabeto, pero se mantiene el número de cadenas reducido y su longitud pequeña, la técnica de Programación Dinámica obtiene las SCS menores.

Para el caso de que el alfabeto se mantenga con pocos símbolos, la longitud de las cadenas sea pequeña y se incremente el número de cadenas del lenguaje, las metaheurísticas como L-M.M. y L-M.M.2, obtienen las SCS de menor longitud.

Cuando se hace una combinación en los lenguajes en el número de cadenas, su longitud y se varía el número de símbolos del que consta el alfabeto, se tiene que la metaheurística Lookahead Function obtiene las SCS de menor longitud.

Si el lenguaje está compuesto por un número muy grande de cadenas de longitud de cientos de caracteres, y un alfabeto pequeño, la metaheurística Lookahead Function es la mejor.

Además, de manera general, las metaheurísticas obtienen las SCS en menor tiempo que la técnica de Programación Dinámica.

Para la técnica de Programación Dinámica, se tuvieron dos opciones, una donde se ordenaba el lenguaje (P.D.Ord.) y otra donde no se ordenaba (P.D.Des.), se concluye que si el lenguaje está compuesto de muchas cadenas, o si su alfabeto contiene muchos símbolos o la longitud de sus cadenas es muy grande o cualquier combinación de las anteriores, se obtiene una SCS de menor longitud si el lenguaje se ordena, en otro caso, con el lenguaje desordenado se encuentra la SCS de menor longitud.

Para el caso de una colonia de quince hormigas y una sola, la diferencia en la longitud en las SCS encontradas por ambas, no es muy notable excepto en el caso de que el alfabeto tenga muchos símbolos, y en cuanto al tiempo, es mucho mayor en la mayoría de los casos para la colonia de quince hormigas, aunque se trate de milisegundos o segundos, lo cual se debe al número de hormigas que se emplea para calcular la SCS.

Se realizaron pruebas con más de una colonia de hormigas, pero el tiempo se elevó demasiado (de minutos a horas y de segundos a minutos) y la longitud de la SCS no disminuyó considerablemente, por lo que se omitieron tales pruebas, sin embargo, el software instrumentado considera éste caso, pero no se recomienda por el crecimiento del tiempo, y este se debe a que se realiza una actualización de la tabla de feromona.

Bibliografía

[1] http://geneura.ugr.es/~jmerelo/tutoriales/heuristics101/

[2] http://www.azc.uam.mx/publicaciones/enlinea2/num1/1-3.htm

[3] http://www.uwasa.fi/cs/publications/2NWGA/node100.html

[4] John E. Hopcroft, Jeffrey D. Ullman. Introducción a la Teoría de Autómatas, Lenguajes y Computación. Compañía Editorial Continental, S.A. de C.V., México.

[5] http://es.wikipedia.org/wiki/Complejidad_computacional

[6] Michael R. Garey, David S. Johnson. Computers And Intractability A Guide To The Theory of NP-Completeness. Bell Laboratories.

[7] Adenso Díaz, Fred Glover, Hassan M. Ghaziri, J.L.González, Manuel Laguna,Pablo Moscato, Fan T. Seng. Optimización Heurística y Redes Neuronales. Editorial Paraninfo S.A., 1996.

[8] http://ingenierias.uanl.mx/9/pdf/ 9_Roger_Rios_et_al_Investigacion_de_oper.pdf

[9] http://www.uwasa.fi/cs/publications/2NWGA/node101.html

[10] http://decsai.ugr.es/~verdegay/dnamica.pdf

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. Introduction to Algorithms. USA: The MIT Press, 2001.

[12] http://www.lcc.uma.es/~av/Libro/CAP5.pdf

[13] David Corne, Marco Dorigo, Fred Glover. New Ideas In Optimization. McGraw-Hill International, 1999.

Apéndice A

Manual de usuario

Esta parte del documento, pretende dar ayuda al usuario acerca del uso del programa que calcula la SCS. Para lo cual se dará una breve explicación sobre cada opción que lo compone, así como de un ejemplo a fin de simplificar su entendimiento.

Para poder ejecutar el programa, se debe de tener una unidad de CD-ROM, Procesador de 166Mhz o superior, 32MB de RAM (128 recomendados), mouse y monitor VGA o superior, sistema operativo Linux kernel 2.2 o superior, sobre las siguientes distribuciones de Linux: Red Hat 6.2, Mandrake 7.2 y SuSE 7.0 o superiores. Al dar doble click sobre el archivo *principal*, que se encuentra en la carpeta SCS del disco, se inicia la aplicación.

A.1. Menú principal.

Una vez que se carga el programa, aparece una pantalla (Figura A-1), la cual es la ventana principal del programa, la que nos permitirá acceder a los diferentes métodos implementados para calcular la SCS, así como a otras opciones. También se muestra un mensaje que desaparece automáticamente después de unos segundos, el cual indica que ésta aplicación debe ser distribuida sobre los términos de la Licencia Pública General GNU, versión 2.

Dentro de la ventana principal, se encuentra un menú que permite elegir varias opciones: *Archivo*, el cual nos permite elegir la técnica que se va a emplear para calcular la SCS, ya sea por medio de la heurística ACO o a través de Programación Dinámica; además de darnos la opción de salir de la aplicación. Otra opción, es la de *Lenguaje*, que nos permite crear un lenguaje,

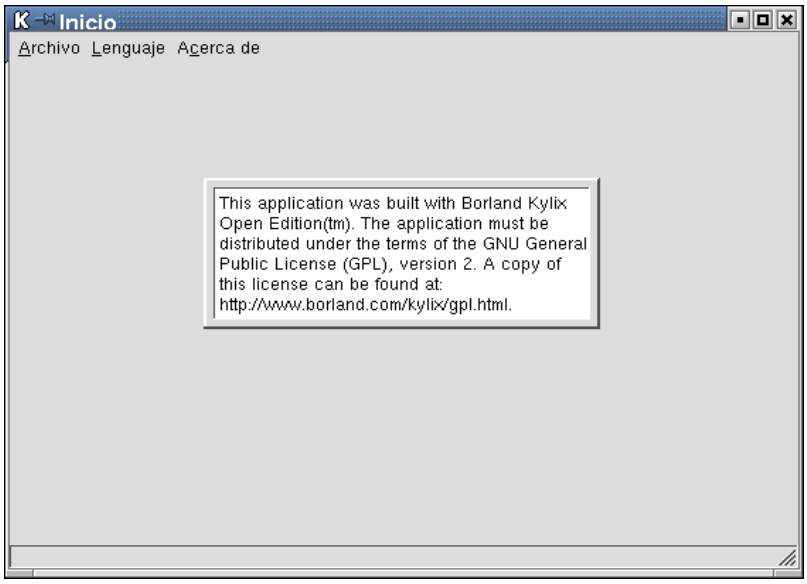

Figura A-1: Ventana que se muestra al ejecutar el programa.

u ordenarlo; y la última opción, es *Acerca de*, donde el usuario puede encontrar información acerca de la aplicación, y ayuda referente a la manera en la que se utiliza la aplicación.

Si se elige el menú *Archivo*, se encontrará con las opciones de *ACO*, *Prog. Din.* y *Salir*. Las dos primeras, nos permiten calcular la SCS de un lenguaje mediante la heurística ACO para el primer caso, y la técnica de Programación Dinámica para el segundo. La tercera opción, sirve para salirnos del programa.

A.2. Heurística ACO.

Si se elige la opción de *ACO*, aparecerá una ventana (Figura A-2). Esta nueva pantalla, contiene dos menús, *Menú* y *Acerca de*. La segunda ya se describió anteriormente, mientras que la primera, se refiere a que se desea calcular la SCS de un lenguaje, ya sea que éste se encuentre en un archivo o se introduzca mediante el teclado; para ello se cuenta con las opciones de menú de *Nuevo* y *Abrir* además de contar con la posibilidad de cerrar ésta ventana y regresar a la principal (*Regresar menú inicio*), o bien, salir del programa (*Salir*).

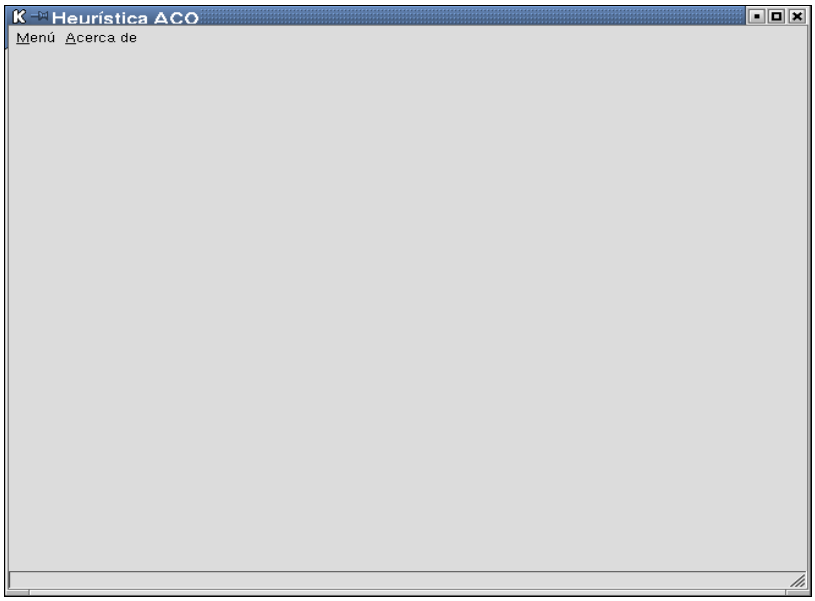

Figura A-2: Ventana principal de la heurística ACO.

A.2.1. Nuevo.

Si se elige del menú la opción *Nuevo*, aparecerán otras opciones acerca de la manera en la que se desea calcular la SCS: *Hallar la SCS*, *Elegir heurística* y *SCS de heurísticas*.

Hallar la SCS.

Si se selecciona la opción *Hallar la SCS*, se calcula la SCS mediante el uso del método de Lookahead Function con una colonia de quince hormigas, la cual, mediante pruebas se encontró que era la que mejor desempeño tenía para obtener la SCS, es decir, la que tenía menor longitud en la mayoría de los problemas. En la ventana, aparecerán campos a fin de indicar el número de cadenas de las que consta el lenguaje para el que se desea obtener la SCS, y los caracteres que componen el alfabeto de dicho lenguaje (Figura A-3).

Si nos olvidamos de introducir algún valor o si éstos son erróneos, es decir, no son enteros positivos, se indica con alguno de los siguientes mensajes de error (Figuras A-4, A-5 y A-6).

Una vez hecho esto, se presiona el botón con la etiqueta OK y aparecen nuevos campos a fin de introducir las cadenas, una en cada línea, que componen el lenguaje (Figura A-7).

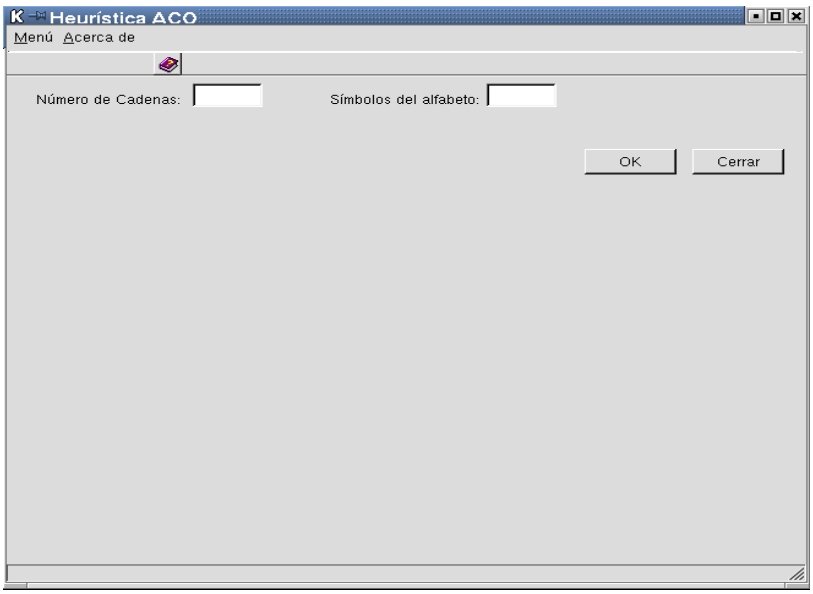

Figura A-3: Ventana donde se introduce el número de cadenas del lenguaje y su alfabeto.

Figura A-4: Error de datos faltantes.

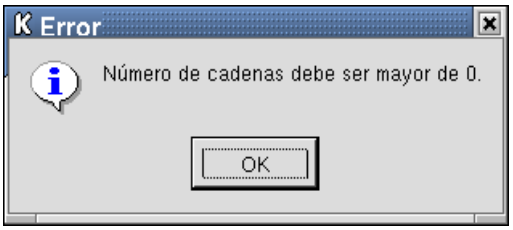

Figura A-5: Error al introducir un dato incorrecto.

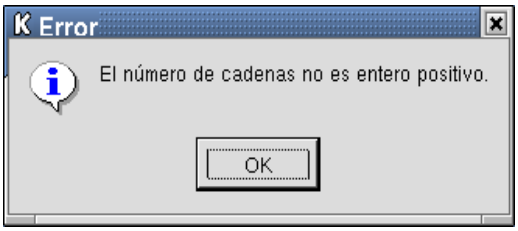

Figura A-6: Error al introducir valores no válidos.

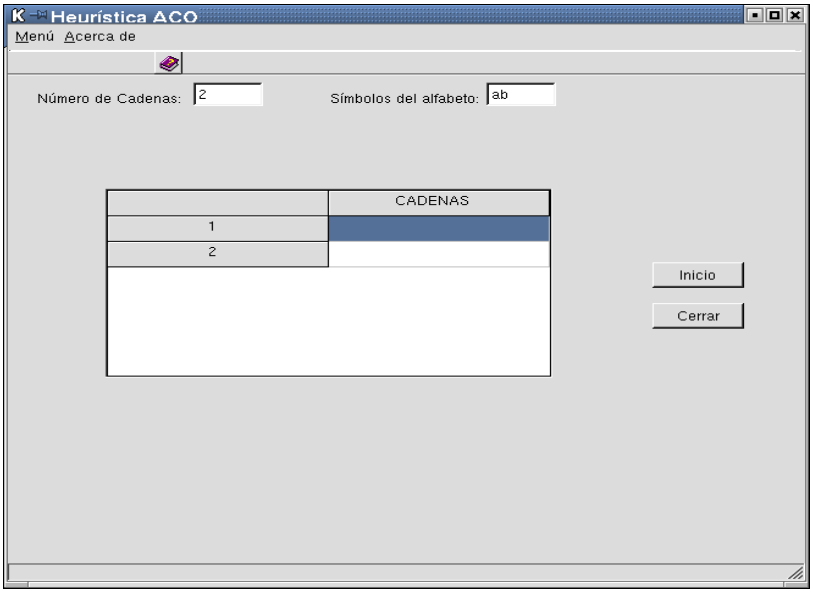

Figura A-7: Ventana que nos permite introducir las cadenas del lenguaje.

Si al introducir una cadena del lenguaje, contiene un símbolo que no esté definido previamente en el lenguaje, se indicará con un mensaje de error (Figura A-8).

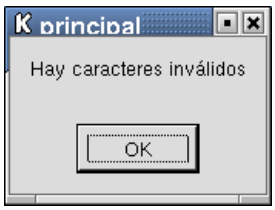

Figura A-8: Error de símbolos en cadenas que no están en el alfabeto.

Al término de introducir las cadenas, al presionar el botón de *Inicio*, se comenzarán los cálculos para obtener la SCS. Cuando se encuentra la SCS, (Figura A-9), aparecen nuevos campos en los que se muestra la SCS así como su longitud, además de dos botones que permiten guardar el lenguaje introducido y/o el resultado obtenido, y un botón para cerrar la ventana de la Heurística ACO y regresar a la ventana principal.

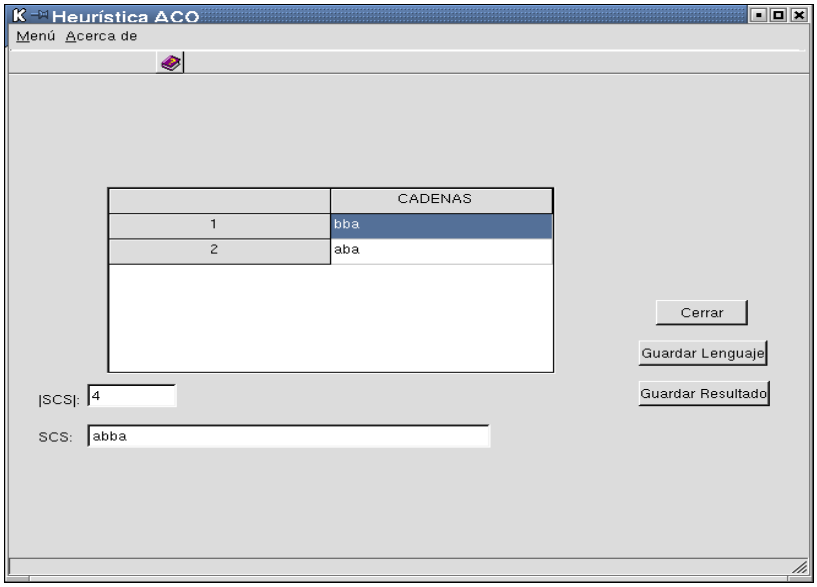

Figura A-9: Ventana donde se muestra la SCS encontrada.

Elegir heurística.

Si se elige la segunda opción, *Elegir heurística*, a diferencia del caso anterior, aparecen campos en los que se permite elegir el número de colonias de quince hormigas con las que cada una de las distintas heurísticas hará el cálculo de la SCS (Figura A-10). Para empezar el cálculo de la SCS, se sigue el mismo procedimiento que en la opción *Hallar la SCS*.

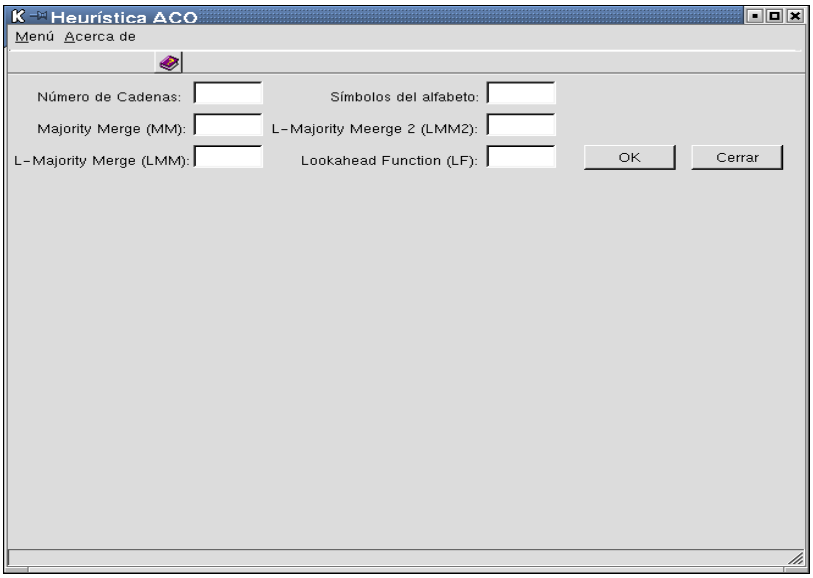

Figura A-10: Ventana para elegir la heurística que obtendrá la SCS.

Si presionamos el botón OK sin introducir ningún valor, aparece un mensaje de error (Figura A-11).

Figura A-11: Error al faltar todos los valores.

Si nos olvidamos de introducir algún valor y presionamos el botón OK, se muestra un

mensaje de error (Figura A-12).

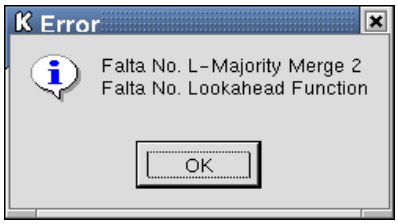

Figura A-12: Error al faltar algún valor.

Si introducimos valores que no sean enteros positivos en cualquiera de los campos, se nos indicará con un mensaje de error (Figura A-13).

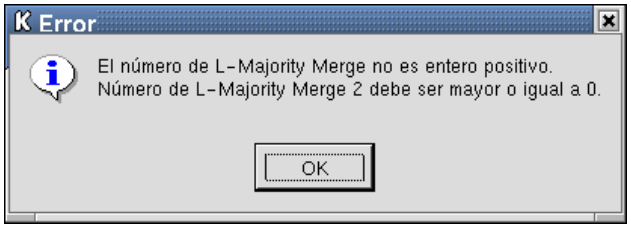

Figura A-13: Error al introducir algún valor de manera inválida.

Una vez introducidos los datos de manera correcta, aparecerán nuevos campos donde introducir las cadenas que forman el lenguaje (Figura A-14).

El resultado, se mostrará de forma distinta que en la opción anterior, ya que aquí se muestra la SCS con menor longitud obtenida por cada una de las diferentes heurísticas, así como la longitud de cada una de ellas (Figura A-15). El orden de los resultados obtenidos por cada técnica son de izquierda a derecha para el caso de su longitud, y de arriba hacia abajo para la SCS encontrada, para M.M., L-M.M., L-M.M.2 y L.F. respectivamente.

SCS de heurísticas.

Si se elige la tercera opción, *SCS de heurísticas*, se sigue el mismo procedimiento que para la opción *Hallar la SCS*, con la diferencia de que a pesar de que no se pide el número de veces que cada heurística realizará el cálculo de la SCS, el resultado muestra la SCS obtenida por cada

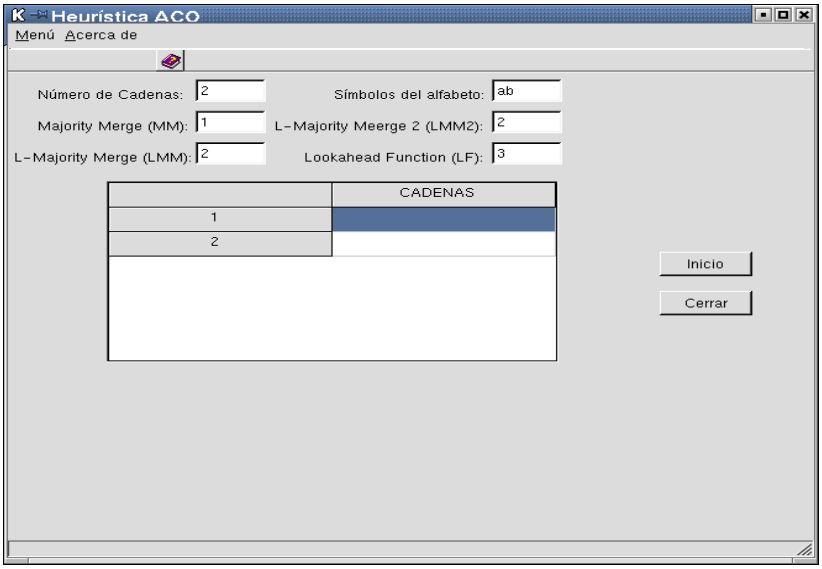

Figura A-14: Ventana para introducir las cadenas.

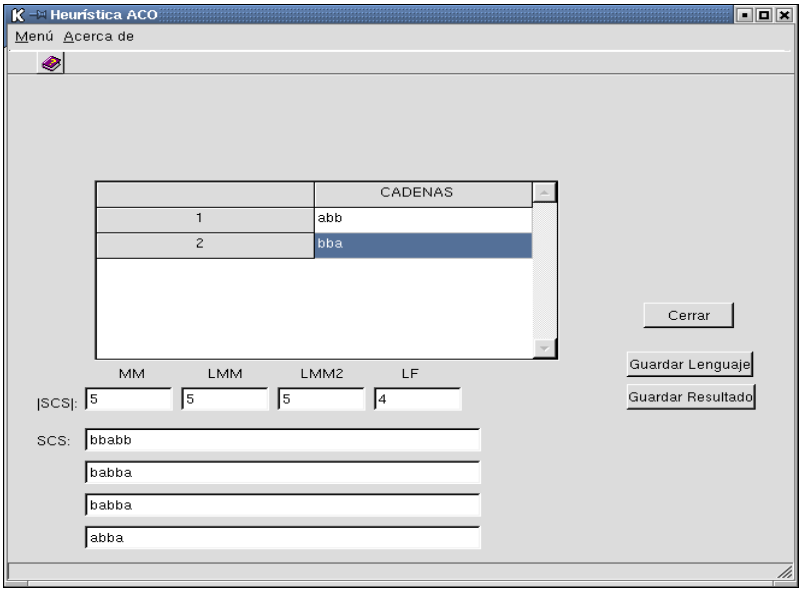

Figura A-15: SCS encontradas para los valores elegidos.

una de las heurísticas empleando una colonia de quince hormigas. Además, se pide el número de cadenas y los símbolos que componen el alfabeto (Figura A-16).

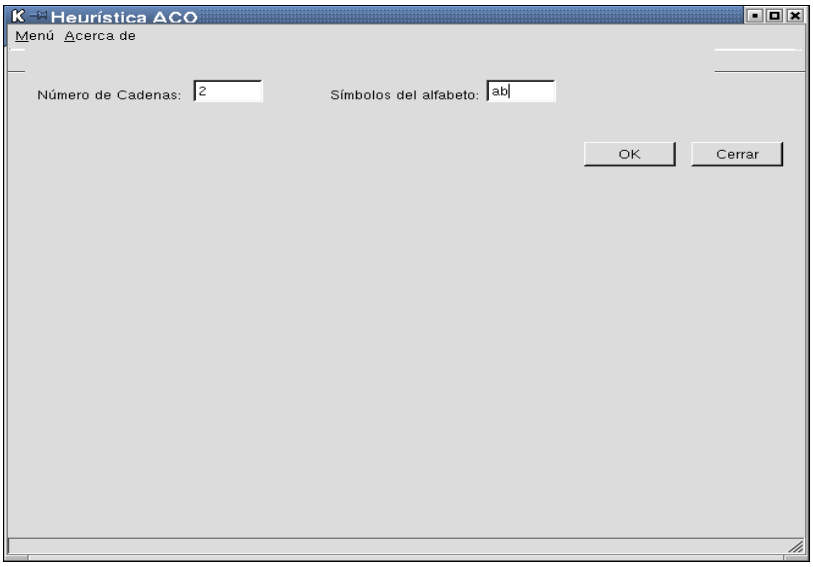

Figura A-16: Ventana principal de SCS de heurísticas.

Si falta algún valor, se muestra un mensaje de error (Figura A-17).

Figura A-17: Mensaje de error de datos faltantes.

Una vez introducido de manera correcta el número de cadenas y los símbolos del alfabeto, aparecen nuevos campos para introducir las cadenas del lenguaje (Figura A-18).

Si hay algún símbolo incorrecto en alguna de las cadenas, se muestra un mensaje de error (Figura A-19).

Una vez que se han introducido las cadenas del lenguaje, y presionar el botón de *Inicio*, se muestra el resultado obtenido (Figura A-20), donde se tienen las SCS menores obtenidas por cada una de las heurísticas.

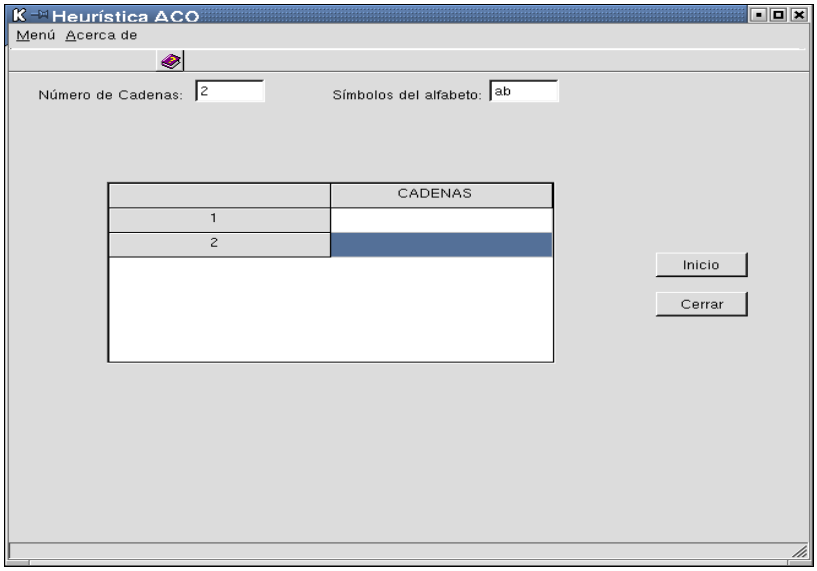

Figura A-18: Ventana para introducir las cadenas que componen el lenguaje.

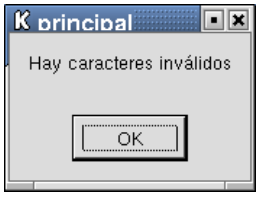

Figura A-19: Mensaje de error para símbolos inválidos.

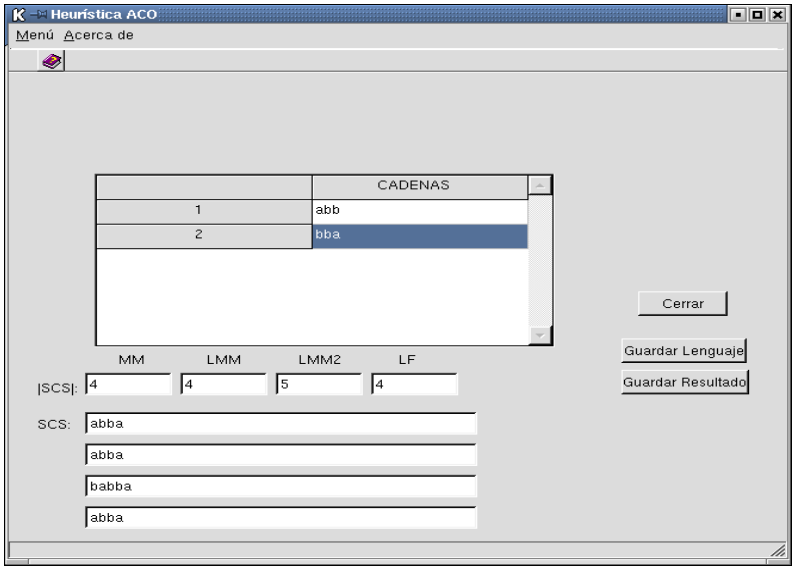

Figura A-20: SCS obtenida por la opción SCS de heurísticas.

A.2.2. Abrir.

Las diferentes opciones del menú *Abrir*, son similares a las del menú *Nuevo*, con la diferencia de que en lugar de que el usuario introduzca las cadenas del lenguaje y el alfabeto, éstos son leídos de un archivo de texto, para lo cual aparece una ventana de abrir (Figura A-21), donde el usuario puede explorar y buscar la ruta donde se encuentra el archivo con el lenguaje del cual se desea calcular su SCS.

Cabe mencionar que aquí no aparece el botón de *Guardar Lenguaje*, ya que éste es leído desde un archivo.

Hallar la SCS.

Una vez elegido el archivo con extensión *.txt, y presionar el botón *Open*, el resultado obtenido se muestra de manera inmediata (Figura A-22).

Elegir heurística.

Para el caso de la opción *Elegir heurística*, se pregunta, como en el caso de *Nuevo*, el número de colonias de quince hormigas con las que cada heurística calculará la SCS (Figura A-23).

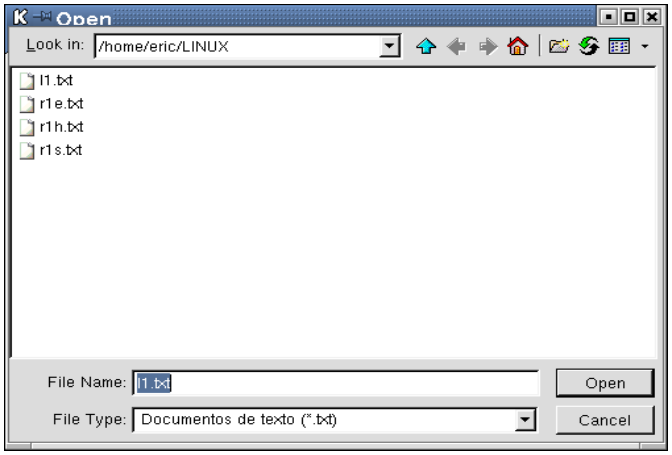

Figura A-21: Abrir un lenguaje de un archivo existente.

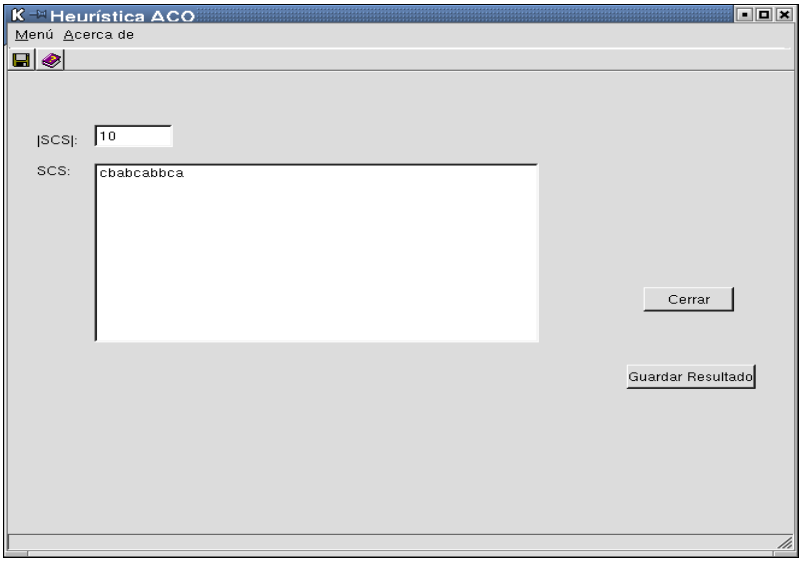

Figura A-22: Resultado obtenido para el lenguaje leído desde archivo.

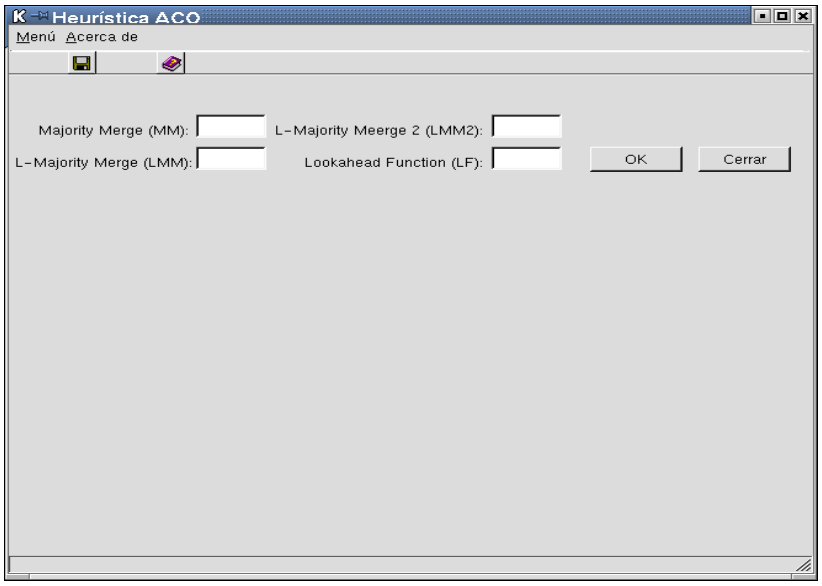

Figura A-23: Opción Elegir heurística de Abrir.

Si existe algún dato incorrecto, se indica con mensajes de error (Figuras A-24 y A-25).

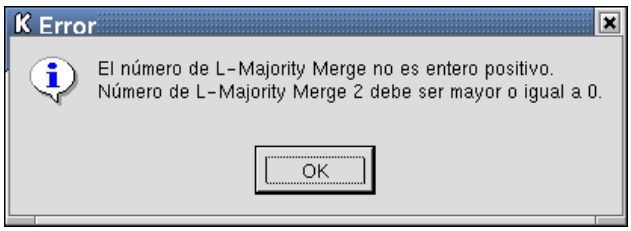

Figura A-24: Si algún valor es letra o entero negativo.

El resultado encontrado se muestra por cada opción elegida (Figura A-26).

SCS de heurísticas.

Aquí no se pregunta el número de colonias de quince hormigas que se emplearán para obtener la SCS, ya que sólo se hace con una colonia de quince hormigas para cada técnica. El resultado es similar al obtenido por la opción anterior (Figura A-27).

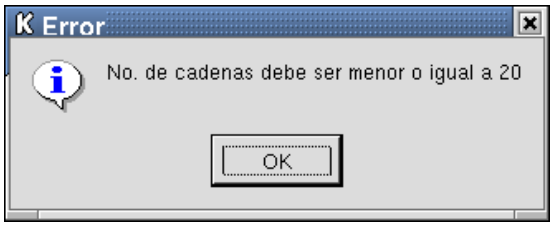

Figura A-25: Si se rebasan lo límites permitidos.

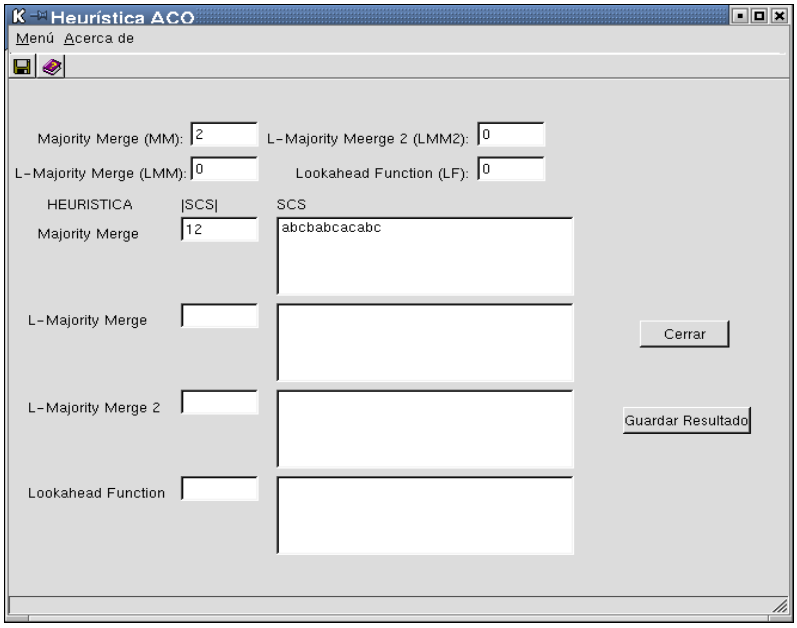

Figura A-26: SCS encontradas para esta opción.

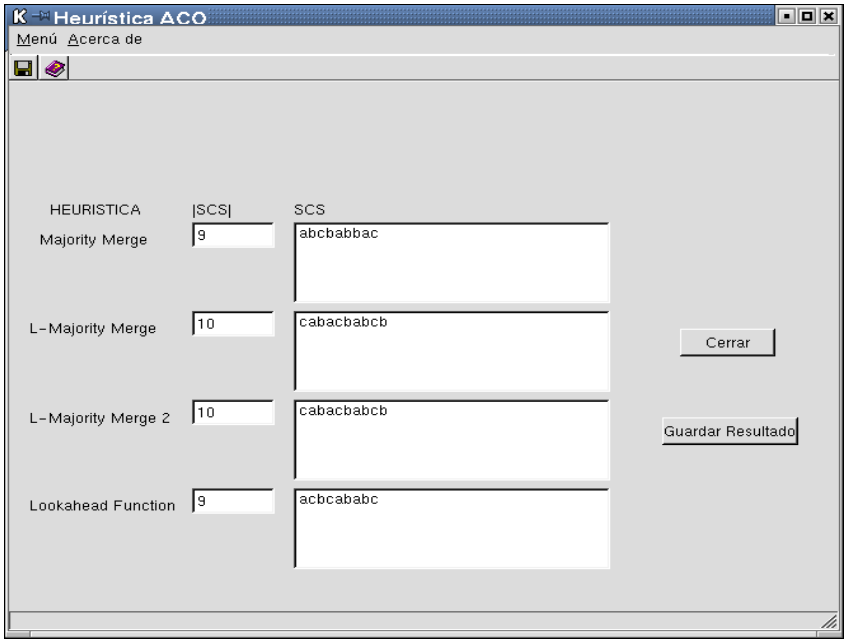

Figura A-27: SCS encontradas por cada metaheurística.

A.3. Programación Dinámica.

Si se elige ésta opción de la ventana principal, aparecerá una ventana (Figura A-28), con un menú que nos permitirá elegir si se obtiene la SCS de un problema leído del teclado o de un archivo de texto, así como de regresar a la pantalla anterior o salir del programa.

A.3.1. Nuevo.

Dentro de ésta opción, se pregunta por el número de cadenas del que consta el lenguaje. Si introducimos algún valor equivocado o faltan valores, se indica con mensajes de error (Figuras A-29 y A-30).

Al presionar el botón de OK, aparecerá una tabla donde introducir cada una de las cadenas. En seguida, se muestra el resultado del problema introducido mediante teclado, al oprimir el botón de Iniciar (Figura A-31).

Si se introduce algún símbolo en alguna cadena que no esté definido en el alfabeto, se indica con un mensaje de error (Figura A-32).

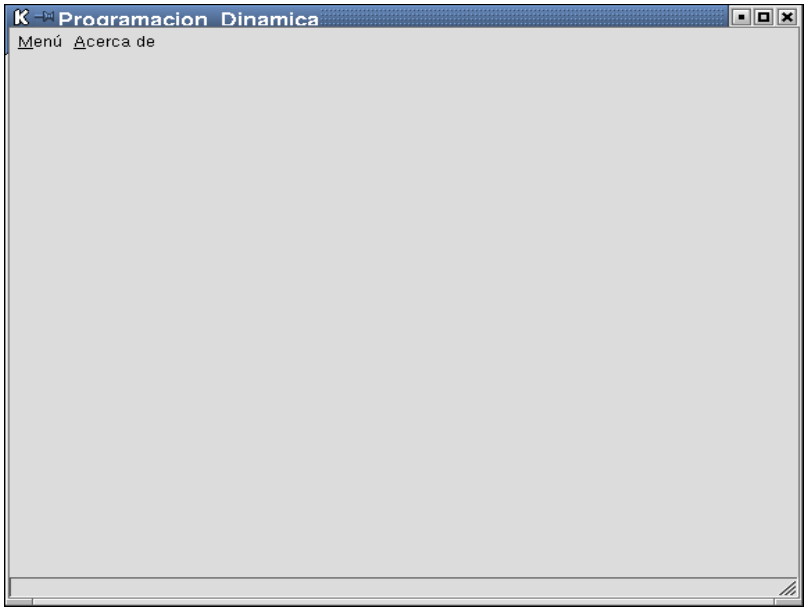

Figura A-28: Ventana principal de la técnica de Programación Dinámica.

Figura A-29: Error al faltar algún dato.

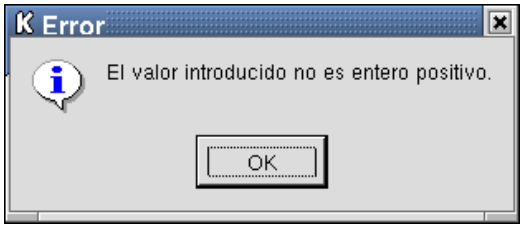

Figura A-30: Error al introducir un valor equivocado.

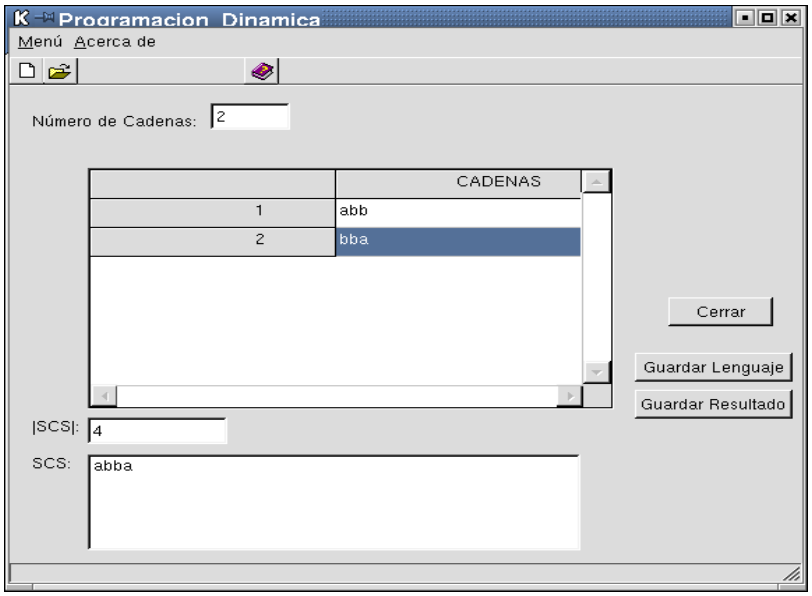

Figura A-31: Ventana donde se muestra el resultado obtenido por Programación Dinámica.

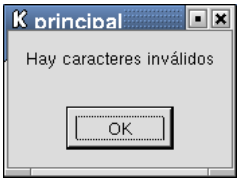

Figura A-32: Error de símbolos incorrectos.

A.3.2. Abrir.

Si se elige esta opción, aparece una nueva ventana (Figura A-33), donde se permite que el usuario busque y seleccione el archivo de texto donde se encuentra el lenguaje del que se quiere encontrar su SCS.

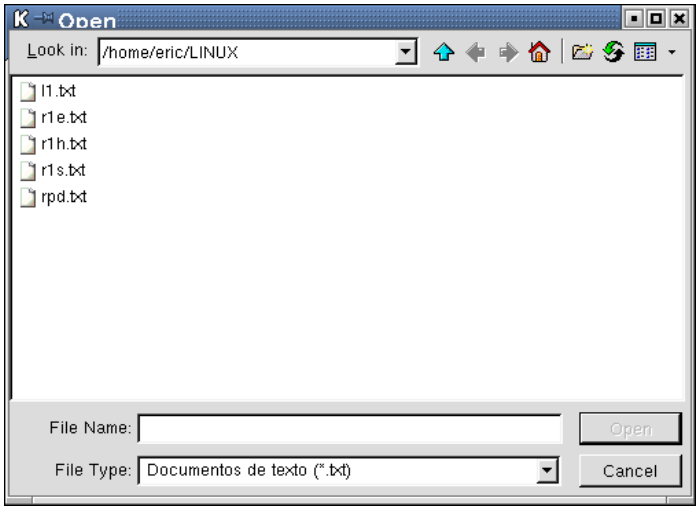

Figura A-33: Ventana donde se elige el archivo que contiene el lenguaje.

El resultado, se muestra en una ventana (Figura A-34), donde se indica la longitud de la SCS y la SCS encontrada mediante la técnica de Programación Dinámica.

A.4. Lenguaje.

Esta opción, nos permite, ya sea crear un lenguaje u ordenarlo.

A.4.1. Crear lenguaje.

Aparece una nueva ventana, (Figura A-35), donde se pide al usuario información acerca del número de cadenas, su longitud, si hay o no incremento y si éste es aleatorio o no. Para guardar el lenguaje, se presiona el botón de Guardar, lo que abre una nueva ventana donde se permite al usuario elegir la ruta donde desea guardarlo, y el nombre del archivo de texto (*.txt).

Si se presiona el botón Guardar, sin introducir ningún valor, se muestra una ventana de

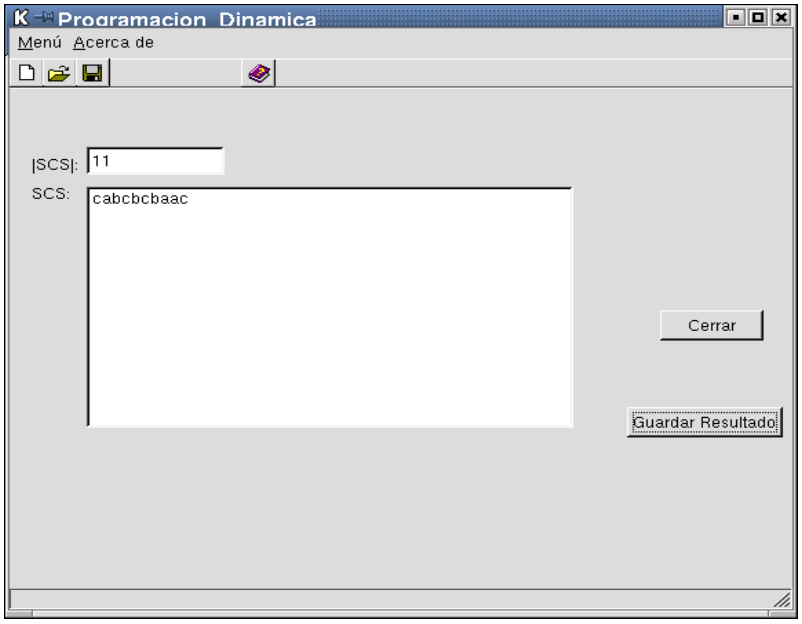

Figura A-34: SCS encontrada por la técnica de la Programación Dinámica de un archivo.

Figura A-35: Ventana para crear un lenguaje.

error (Figura A-36).

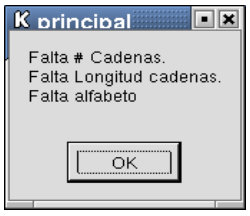

Figura A-36: Error al faltar datos.

En la tabla, se introduce el número de cadenas del que consta el lenguaje, así como su longitud. También, se indican los símbolos del alfabeto (Figura A-37).

Figura A-37: Ventana donde se introducen las características del lenguaje a crear.

El incremento, se refiere a si la longitud es fija, o a partir de la longitud se incrementa en un cierto valor.

Si no seleccionamos el incremento, se muestra una pantalla (Figura A-38).

En caso de que exista un incremento, si no es aleatorio, se incrementa en una cantidad constante. En caso de ser aleatorio, se especifican los límites en los que el incremento se elige de manera aleatoria en ese rango.

Se puede tener hasta 5 estilos de cadenas diferentes en un mismo lenguaje (Figura A-39).

Si se selecciona el incremento y no se elige *Aleatorio*, tenemos un mensaje de error (Figura A-40).

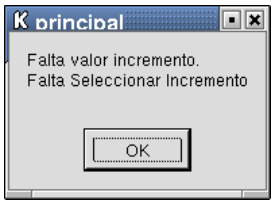

Figura A-38: Errores de algunos valores equivocados.

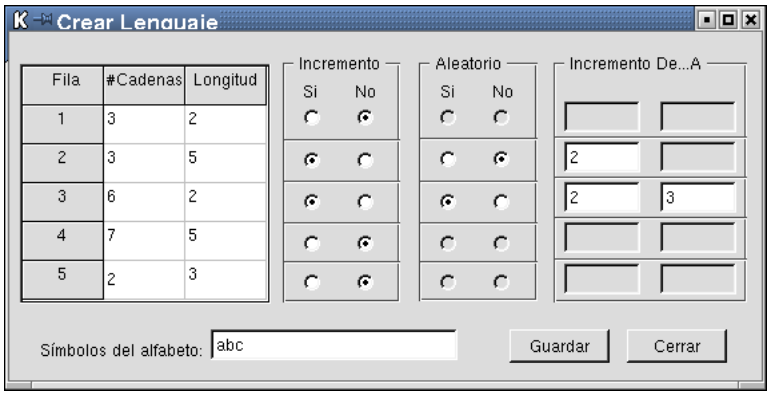

Figura A-39: Diferentes estilos de cadenas para un mismo lenguaje.

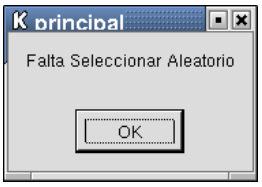

Figura A-40: Error al faltar indicar valor de incremento.

Al determinar los valores del lenguaje, y presionar el botón *Guardar*, aparece una ventana donde se indica el nombre y la ruta en donde se guardará el lenguaje (Figura A-41).

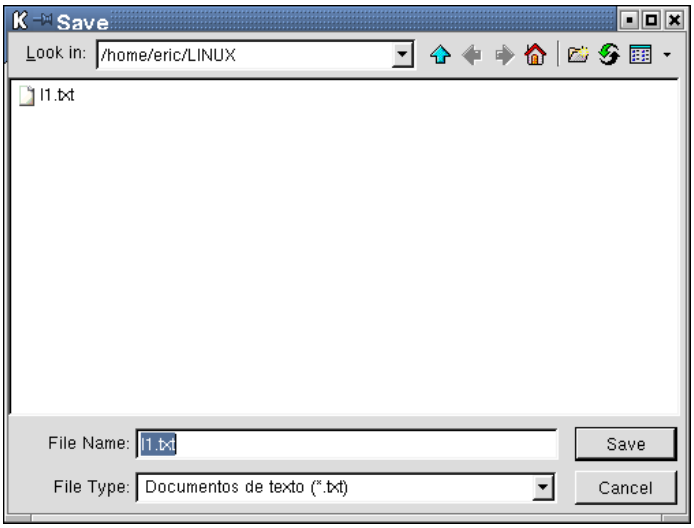

Figura A-41: Ventana para elegir el archivo donde guardar el lenguaje creado.

A.4.2. Ordenar lenguaje.

Este opción nos permite ordenar un archivo de texto existente que contiene un lenguaje, a fin de que se pueda obtener una SCS de menor longitud (Figura A-42), donde se puede seleccionar el archivo a ordenar. Cabe mencionar que se sobrescribe el archivo ordenado sobre si mismo.

A.5. Acerca de.

En el menú *Acerca de*, es donde el usuario puede encontrar información acerca de la aplicación (*Acerca de*), y ayuda (*Ayuda*) sobre la manera en la que se utiliza la aplicación (Figura A-43).

A.6. Formato del lenguaje en un archivo.

Si se desea crear un lenguaje mediante un procesador de texto, el formato es el siguiente: se escribe una cadena, se separa de otra por medio de un espacio en blanco, y así sucesivamente,

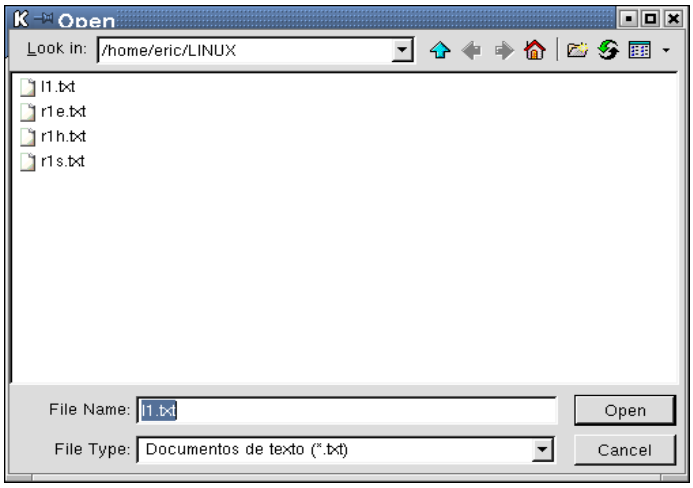

Figura A-42: Ventana para seleccionar el archivo a ordenar.

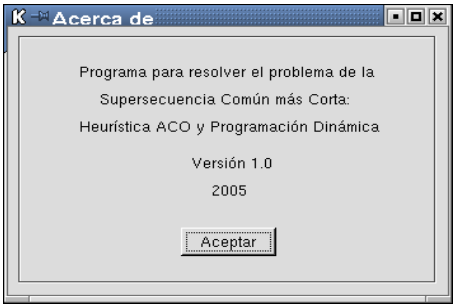

Figura A-43: Ventana con información de Acerca de.

hasta terminar de escribir todas las cadenas. Al final, debe de quedar un espacio en blanco. Además, éste deberá de ser guardado con la extensión *.txt. Un ejemplo, se muestra en el apéndice C.

A.7. Guardar resultado y lenguaje.

Estas dos opciones aparecen en la mayoría de las ventanas, y se describen a continuación.

En *Guardar lenguaje*, podemos guardar el lenguaje que se introdujo de forma manual, para el cual se obtuvo la SCS, y en *Guardar resultado*, nos permite guardar el resultado obtenido (Figura A-44), donde se escribirá el nombre del archivo. Cabe mencionar que el archivo tiene la extensión *.txt.

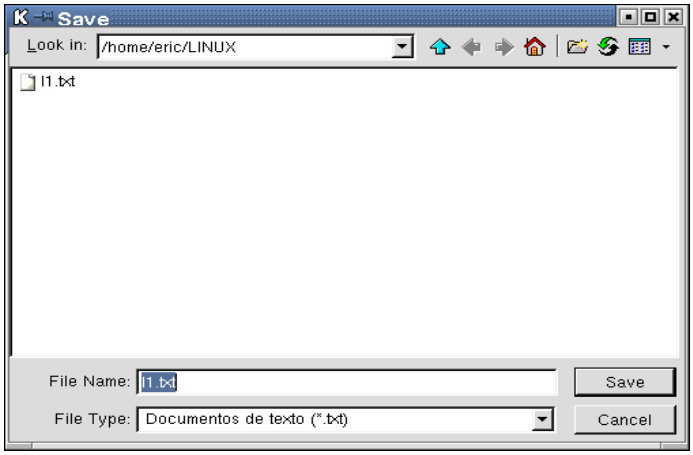

Figura A-44: Ventana que nos permite guardar el resultado de la SCS encontrada.

Apéndice B

Ejemplos

En éste apéndice, se muestran los ejercicios que se realizaron de manera manual, el resultado obtenido y su comparación con las SCS obtenidas con la aplicación realizada. A continuación se describe cada uno de los lenguajes sobre los que se hicieron las pruebas.

$L1 = \{abc, bbca, cbabac, cbab, abcba\}$.

$L2 = \{cb, bc, abc, cba\}$.

$L3 = \{abb, ab, cbab, cccabb, cabacbabb, ababcab, cbabbca\}$.

$L4 = \{abab, bcbab, cbabcba, abccb, cbabccbab, cbabbba, abcbab, cbabbc, cbabcba, abcbacb, abcbabcba, bbbabbcb, abcbcbccbaaa, aabcbacbbac\}$.

$L5 = \{ab, ba, cba, cbabac, abcba, cbabc, abcabb\}$.

$L6 = \{bcba, bbbaacb, babcab, bbaccaab, bcbbabac\}$.

$L7 = \{ccc, bbb, aaa, aabbc, bbcca, ccaab, babac, cbabc, abc\}$.

$L8 = \{ab, bbbaa, bba, bccbba, bacbabbcca, bcbabbaccb, bcbacbab, cbab, bbbaacb, bbaabbcc, aaccbbab\}$.

$L9 = \{abc, cccbbb, abcbacbabb, bbbccccabbbaabcb, cbaabcbb, bbbbcccaaababaaa, cbabcbabcbc, cbacbabbcbbcbaa, abcbb, bcaabcbabbbc, ababbbcba, accb, bcca\}$.

$L10 = \{abab, bcbab, cbabcba, abccb, cbabccbab, cbabbba, abcbab, cbabbc, cbabcba, abcbacb, abcbabcbab, bbbabbccb\}$.

$L11 = \{abcb, bccbabcba, bcbcbabccbb, cbabbcbaccbac, ccbabb, bccbbbcca, abbcbabc, bcbbacbbbabcba, abcbbbcb, bcccbbaaa, aaabcbcbac, bcbcabcbcbac\}$.

$L12 = \{ab, ba\}$.

$L13 = \{ababba, aabbaaba\}$.

$L14 = \{aababaa, babaaaa\}$.

$L15 = \{bababbb, babbbbbbab\}$.

$L16 = \{abababaabaababababaaab, aaabaaaaabaaabaabbb, aaaabbaaaababbabaa,$
$aaababbbaabaaaaba, baaaaaaabbbbbabb, aabbaabbaaabaab, bbabbbaabbaabb,$
$abbbbbbaaabba, babbbabbbaaa, abbaaabaabb, abbababab\}$.

Los resultados se muestran en la siguiente tabla:

Lenguaje	Manual	Prog. Din.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	**8**	**8**	**8**	9	9	9
L2	**5**	**5**	**5**	**5**	**5**	**5**
L3	**11**	**11**	**11**	**11**	**11**	**11**
L4	**18**	**18**	**18**	**18**	19	19
L5	**9**	**9**	**9**	**9**	10	10
L6	**12**	14	**12**	13	13	13
L7	**10**	12	11	12	12	11
L8	**16**	**16**	**16**	**16**	17	18
L9	**27**	**27**	31	32	**27**	28
L10	**14**	15	16	15	**14**	15
L11	**21**	**21**	**21**	**21**	**21**	**21**
L12	**3**	**3**	**3**	**3**	**3**	**3**
L13	**9**	**9**	**9**	10	**9**	**9**
L14	**9**	**9**	**9**	**9**	**9**	**9**
L15	**11**	**11**	**11**	**11**	**11**	12
L16	32	**28**	**28**	**28**	**28**	**28**

Tabla 31. Longitudes de las SCS encontradas.

Además de los ejemplos anteriores, se realizaron pruebas con lenguajes que tuvieran longitudes más grandes, por lo que sólo se hicieron pruebas para la aplicación creada, ya que de hacerlas manualmente, se tardaría mucho tiempo. A continuación se describen los lenguajes.

L17: 20 cadenas, empezando de 100 símbolos hasta cinco, $\Sigma = \{a, b, c, d\}$.

L18: 16 cadenas de longitud 160, $\Sigma = \{a, b, c, d, e, f, g, h\}$.

L19: 4 cadenas de longitud 80 y 4 de longitud 40, $\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r\}$.

L20: 7 cadenas de longitud 100, $\Sigma = \{a, b, c, d, e, f, g, h\}$.

L21: 23 cadenas de longitud 13, $\Sigma = \{a, b, c, d\}$.

L22: 40 cadenas de longitud 13, $\Sigma = \{a, b\}$.

A continuación se muestran las longitudes de la SCS obtenidas para cada lenguaje.

Lenguaje	Prog. Din.	L.F.	M.M.	L-M.M.	L-M.M.2
L17	**192**	208	212	194	194
L18	633	**595**	656	617	623
L19	**256**	349	341	312	308
L20	**291**	300	327	318	319
L21	43	**37**	39	**37**	**37**
L22	26	**22**	**22**	**22**	**22**

Tabla 32. Longitudes de las SCS encontradas para cada lenguaje.

Otras lenguajes que se utilizaron para hacer las pruebas y comparar los resultados obtenidos por las diferentes técnicas se tienen en los siguiente grupos:

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\},$$

L1: 100 cadenas de longitud 100.

L2: 100 cadenas de longitud 10.

L3: 100 cadenas de longitud 50.

L4: 100 cadenas de longitud 150.

L5: 100 cadenas de longitud 200.

L6: 100 cadenas de longitud 500.

L7: 10 cadenas de longitud 100.

L8: 50 cadenas de longitud 100.

L9: 150 cadenas de longitud 100.

L10: 200 cadenas de longitud 100.

L11: 500 cadenas de longitud 100.

L12: 50 cadenas de longitud 100.

L13: 10 cadenas de longitud 200.

L14: 10 cadenas de longitud 500.

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j\},$$

L15: 10 cadenas de longitud 500.

L16: 50 cadenas de longitud 300.

L17: 50 cadenas de longitud 50.

$$\Sigma = \{a, b, c, d, e, f, g, h\},$$

L18: 200 cadenas de longitud 20.

L19: 500 cadenas de longitud 50.

$$\Sigma = \{a, b, c, d, e\},$$

L20: 100 cadenas de longitud 100.

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\},$$

L21: 100 cadenas de longitud 100.

$$\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u\},$$

L22: 100 cadenas de longitud 100.

B.1. Una colonia de 15 hormigas.

Para este caso, las longitudes de las SCS, se muestran a continuación:

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	1436	1461	**1252**	1326	1273	1270
L2	175	182	**154**	169	163	159
L3	742	755	**645**	708	661	662
L4	2120	2160	**1835**	1955	1879	1901
L5	2821	2855	**2442**	2576	2504	2498
L6	6884	6946	**5979**	6260	6151	6151
L7	**489**	532	663	739	691	679
L8	1169	1169	**1109**	1200	1165	1151
L9	1596	1583	**1312**	1375	1351	1366
L10	1642	1638	**1344**	1399	1372	1365
L11	1851	1846	**1428**	1475	1447	1444

Tabla 33. Longitudes de las SCS.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L12	1164	1181	**1104**	1185	1128	1149
L13	**968**	1055	1265	1373	1303	1310
L14	**2400**	2560	3168	3356	3361	3365
L15	**1755**	1836	1902	2060	2027	2047
L16	1802	1805	**1512**	1600	1579	1583
L17	325	314	**275**	285	290	282
L18	140	138	114	114	**112**	116
L19	344	331	265	265	**263**	**263**
L20	401	393	**323**	347	328	329
L21	962	966	**788**	825	811	818
L22	1241	1238	**1038**	1108	1065	1073

Tabla 34. Longitudes de las SCS (continuación).

Los tiempos, que tardaron cada una de las técnicas, se muestran en la siguientes tablas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	16:498	15:894	19:916	1:362	**1:162**	1:361
L2	223	229	2:559	140	**128**	143
L3	3:015	3:113	11:323	628	**530**	626
L4	46:724	47:295	32:089	**1:307**	1:940	2:308
L5	1:43:513	1:45:527	42:834	3:426	**2:908**	3:578
L6	23:11:247	23:38:596	1:46:542	14:112	**11:903**	13:196
L7	331	411	8:470	275	202	**125**
L8	5:900	4:900	21:575	832	**619**	676
L9	29:483	29:217	20:245	1:921	**1:811**	2:118
L10	41:561	42:342	20:826	2:451	**2:386**	2:807
L11	2:17:234	2:14:141	24:505	5:949	6:085	7:191

Tabla 35. Tiempos en minutos, segundos y milisegundos para este caso.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L12	5:898	6:120	22:159	827	**585**	672
L13	2:260	2:709	17:796	633	436	**321**
L14	30:605	36:505	48:114	2:371	1:793	**458**
L15	25:341	28:640	7:851	1:434	971	**958**
L16	1:07:245	1:10:121	5:460	1:787	**1:610**	1:734
L17	624	617	816	124	**102**	132
L18	527	508	283	**105**	116	138
L19	7:985	7:592	1:353	**755**	832	1:001
L20	3:870	4:846	567	**378**	395	432
L21	11:176	11:335	4:680	744	**686**	824
L22	14:298	13:580	11:197	1:052	**941**	1:088

Tabla 36. Tiempos en minutos, segundos y milisegundos (continuación).

B.2. Una sola hormiga.

Las longitudes de las SCS encontradas se muestran en las siguientes tablas.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	1436	1461	**1267**	1323	1300	1270
L2	175	182	157	164	**156**	159
L3	742	755	**647**	712	673	662
L4	2120	2160	**1865**	1975	1901	1901
L5	2821	2855	**2452**	2595	2503	2498
L6	6884	6946	**5993**	6328	6178	6151
L7	**489**	532	681	739	676	679
L8	1169	1169	**1148**	1206	1147	1151
L9	1596	1583	**1334**	1398	1366	1366
L10	1642	1638	**1353**	1432	1365	1365
L11	1851	1846	1447	1467	**1444**	**1444**

Tabla 37. Longitudes de las SCS para este experimento.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L12	1164	1181	**1128**	1190	1146	1149
L13	**968**	1055	1344	1707	1327	1310
L14	**2400**	2560	3176	3410	3340	3365
L15	**1755**	1836	1928	2071	2041	2047
L16	1802	1805	**1527**	1617	1580	1583
L17	325	314	**266**	291	282	282
L18	140	138	113	114	**112**	116
L19	344	331	265	265	**263**	**263**
L20	401	393	**323**	329	330	329
L21	962	966	**790**	832	818	818
L22	1241	1238	**1051**	1127	1079	1073

Tabla 38. Longitudes de las SCS para este experimento (continuación).

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L1	16:498	15:894	1:176	115	93	103
L2	223	229	211	9	7	8
L3	3:015	3:113	780	55	49	41
L4	46:724	47:295	2:070	182	157	173
L5	1:43:513	1:45:527	2:800	276	237	255
L6	23:11:247	23:38:596	7:506	1:183	1:048	1:081
L7	331	411	666	19	14	9
L8	5:900	4:900	1:479	9	45	47
L9	29:483	29:217	1:405	150	**138**	156
L10	41:561	42:342	1:477	193	**178**	201
L11	2:17:234	2:14:141	1:724	**443**	444	502

Tabla 39. Tiempo en minutos, segundos y milisegundos utilizado por cada técnica.

Lenguaje	P.D.Des.	P.D.Ord.	L.F.	M.M.	L-M.M.	L-M.M.2
L12	5:898	6:120	1:465	72	54	**47**
L13	2:260	2:709	1:275	48	**33**	36
L14	30:605	36:505	3:231	205	149	**124**
L15	25:341	28:640	510	129	94	**91**
L16	1:07:245	1:10:121	415	171	**159**	165
L17	624	617	52	9	**7**	9
L18	527	508	21	8	**7**	10
L19	7:985	7:592	111	**74**	76	85
L20	3:870	4:846	49	36	39	38
L21	11:176	11:335	331	60	55	72
L22	14:298	13:580	832	85	76	85

Tabla 40. Tiempo en minutos, segundos y milisegundos utilizado por cada técnica (continuación).

Apéndice C

Formato del lenguaje en un archivo.

C.1. Formato del lenguaje en un archivo.

El formato del lenguaje leído de un archivo, ya sea que se genere mediante la aplicación o se escriba de forma manual, debe de escribirse de la siguiente manera: una cadena separada de la siguiente por un espacio en blanco y así sucesivamente, hasta que se termina de introducir todas las cadenas, y el final debe de haber un espacio en blanco (Figura C-1). La extensión del archivo debe de ser *.txt para que pueda ser leído correctamente por la aplicación.

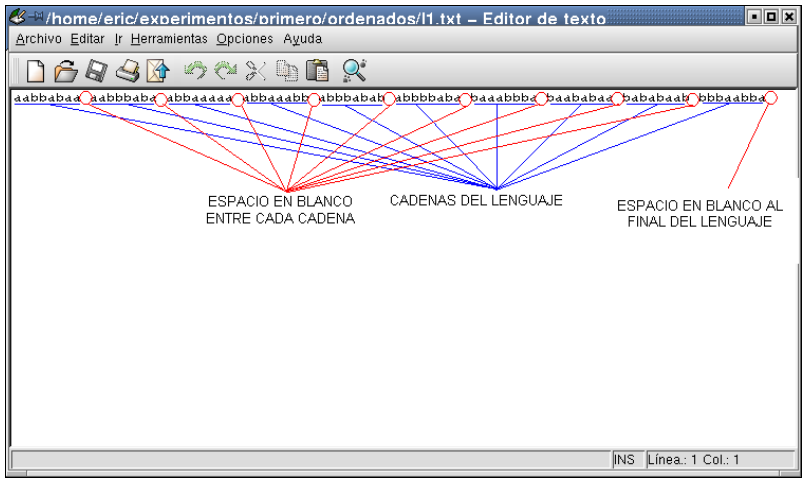

Figura C-1: Formato del archivo que contiene el lenguaje a leer.

A continuación, se muestran dos archivos, uno desordenado (Figura C-2) y otro ordenado

(Figura C-3). El orden en caso de que sea ordenado, es de mayor a menor longitud de la cadena, y en caso de haber dos o más con la misma longitud, se ordenan lexicográficamente.

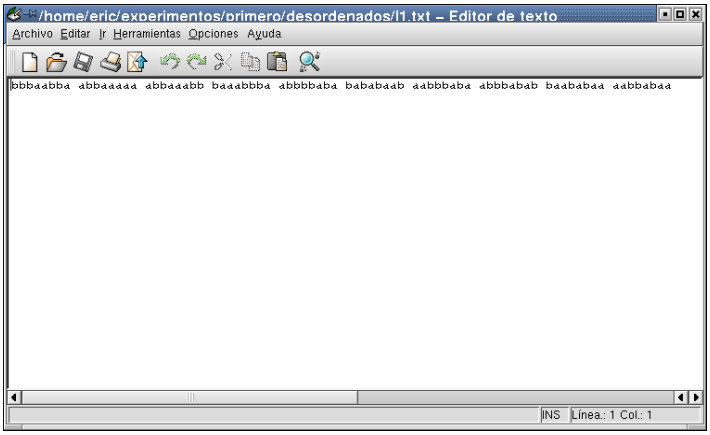

Figura C-2: Archivo que contiene un lenguaje desordenado.

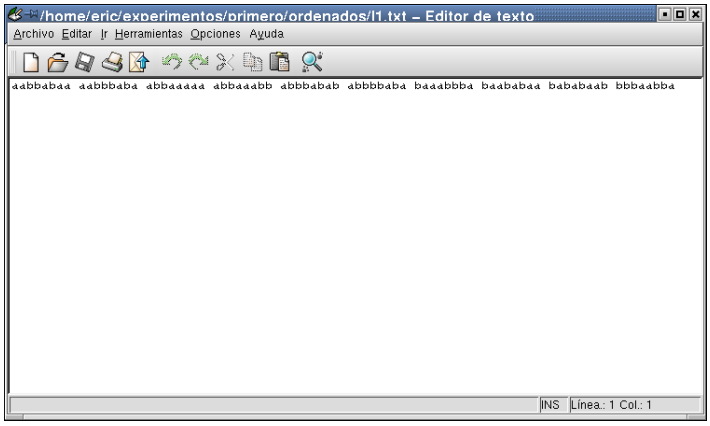

Figura C-3: Archivo que contiene un lenguaje ordenado.

C.2. Archivo de salida.

A continuación se muestran los diferentes archivos de salida para cada técnica.

C.2.1. Heurística ACO.

Hallar la SCS (Figura C-4).

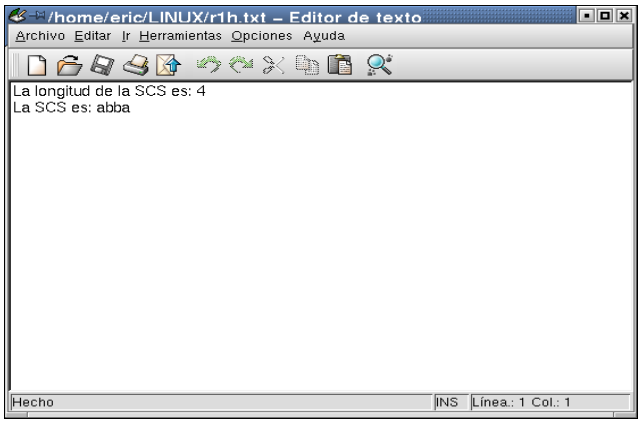

Figura C-4: Archivo de salida para Hallar la SCS.

Elegir SCS (Figura C-5).

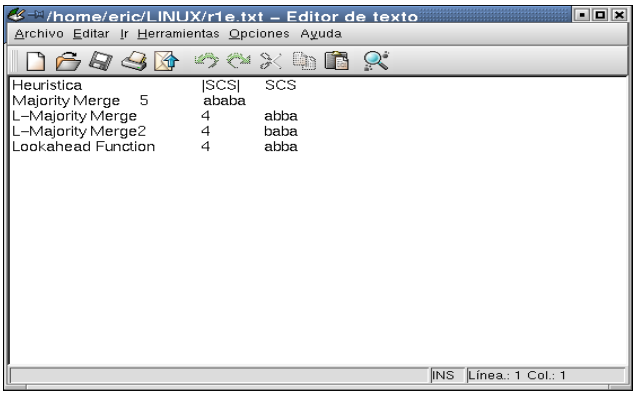

Figura C-5: Archivo de salida para Elegir Heurística.

SCS de heurísticas (Figura C-6).

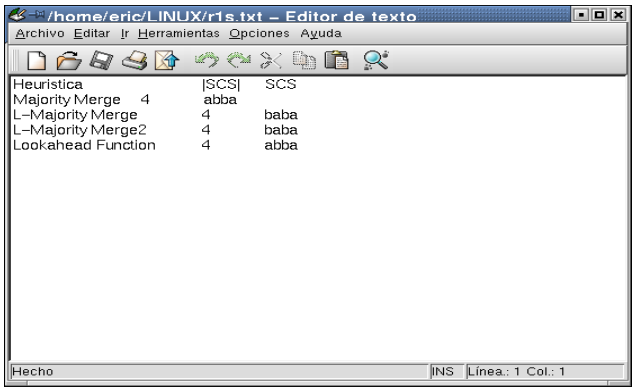

Figura C-6: Archivo de salida para SCS de Heurísticas.

C.2.2. Programación Dinámica.

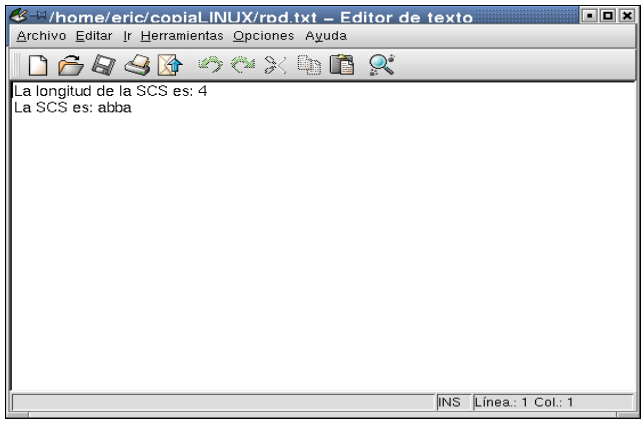

Figura C-7: Archivo de salida para Programación Dinámica.

Apéndice D

Glosario

Algoritmo. Un algoritmo es una serie finita, inequívoca e inambigüa de pasos para resolver un problema.

Computable. Que se puede computar. Determinar indirectamente [una cantidad, esp. el tiempo] por el cálculo de ciertos datos.

Feromona. Sustancia producida por las glándulas sexuales de algunos animales para atraer al individuo del otro sexo. Viene del griego y significa "llevo excitación".

Instancia. Conjunto de datos prueba.

Máquina determinista. Es aquella en que para una misma entrada, siempre se obtiene el mismo resultado.

Máquina no-determinista. Es aquella que para una misma entrada, se pueden obtener resultados diferentes.

Mecánica estadística. Es la aplicación de la estadística, que incluye herramientas matemáticas para tratar con grandes poblaciones en el campo de la mecánica, lo que concierne al movimiento de las partículas u objetos sujetos a una fuerza. Suministra una base para relacionar las propiedades microscópicas de los átomos y moléculas individuales a las correspondientes macroscópicas de los materiales, las que pueden ser observadas en la vida diaria, explicando la termodinámica como un resultado natural de la estadística y la mecánica (clásica y cuántica).

Prefijo. Partícula que se antepone a una palabra para modificar su sentido o para formar otra palabra. Está formado por los primeros símbolos de una cadena.

Psicoanálisis freudiano. Es el estudio de los elementos que integran el psiquismo. Con-

stituye una teoría general del comportamiento humano, que se reduce a las tensiones entre el principio del placer y el principio de realidad, que constantemente se opone al placer. Lo que originariamente surgió como método de investigación y terapia de las neurosis, se convirtió progresivamente en teoría general, no sólo del comportamento humano, sino de la misma naturaleza del hombre y de sus manifestaciones fundamentales.

Satisfactibilidad. Problema de decidir si existe una asignación de valores de verdad para los átomos de una fórmula proposicional que la hagan verdadera (satisfactible).

Sinergia. Acción de dos o más causas cuyo efecto es superior a la suma de los efectos individuales.

Stigmergia. Significa colaboración a través del medio físico. En sistemas descentralizados, tales como las colonias de hormigas, los diferentes componentes colaboran a través de pautas o hitos dejados en el medio: feromonas, acumulación de objetos, o cualquier otro tipo de cambio físico, como la temperatura.