Universidad Carlos III de Madrid
Escuela Politécnica Superior
Ingeniería de Telecomunicación

Proyecto Fin de Carrera

Influencia de las transformaciones lineales en algoritmos genéticos híbridos para el Problema de Steiner

Autor: Ramón Casero Cañas

Tutores: Dr. Sancho Salcedo Sanz

Dr. Carlos Bousoño Calzón

Febrero 2003

Primera persona singular
del condicional simple,

Yo a_ María.

Agradecimientos

Me gustaría romper el orden tradicional de los agradecimientos –primero los académicos y luego los personales– con un recuerdo para mi abuelo Ramón y mi tío Pedro; fueron dos hombres geniales a quienes toda la familia quisimos muchísimo. También para mi excepcional abuela Rosario, para que nos siga haciendo felices mucho tiempo.

Quiero agradecerle a mis tutores Sancho y Carlos (además, enhorabuena) su apoyo y, sobre todo, paciencia. En mi descargo, debo decir que yo también he necesitado en ocasiones bastante paciencia con ellos `:-)` . Es un camino largo el que hemos recorrido juntos, y no puedo imaginar mejores compañeros de viaje. Quiero darle en concreto las gracias a Carlos por soportarme desde 2º, y haberme alentado a investigar pasando por alto mis errores, y a Sancho porque sin su atención diaria este Proyecto no habría sido posible.

A Marc Schoenauer, por su colaboración esencial en el manejo de la biblioteca EO, y la atención dedicada por correo-e y en Madrid. A Franz Rothlauf, por mandarme su libro y aclarar mis dudas.

A Juan Manuel Molera, por su ayuda con el álgebra lineal, tan importante para el Proyecto.

A Bill Allombert y Karim Bellabas por sus consejos y el desarrollo de la biblioteca PARI.

A Xianping Ge por su colaboración en el recuento de ciclos en las primeras fases del Proyecto.

A Pepe, por sus interesantes conversaciones sobre algoritmos genéticos, indicaciones y correcciones, su buen humor y apoyo.

Al servicio de consultas de la RAE (Real Academia Española), por aclarar algunas de mis dudas sobre nuestro idioma.

A esas otras personas que colaboraron de un modo u otro con el Proyecto, David Larrabeiti, Fran Valverde, Arturo de Pablo y Juan Julián Merelo Guervós.

En general, a todos los profesores y profesoras que se olvidan de cerrar la puerta.

También a Saúl y Juanma, por su ayuda con la red del departamento.

Y si hasta aquí nos hemos ocupado de la mente, gracias Irene López por arreglarme la espalda.

Quiero agradecerle a mis padres su amor y apoyo todos estos años, especialmente en los momentos duros.

También a mis numerosos tíos, tías, primos y primas en distintos grados, por lanzarme por los aires, cuidarme, enseñarme tantas cosas y haber tenido y tener siempre tiempo para mí.

A Jaime. Darle las gracias un poco por todo, aunque me gustaría destacar nuestros viajes, y cuando fuimos avenívoros.

A mis mejores amigos, Javier, César, Manuel, Martín y Samuel, porque con gente así da gusto; espero que sigamos disfrutando juntos de tantas cosas. A Mar y Macarena por crear la burbuja y alejar a las chicas.

A Eva, por darme asilo, porque hemos hablado y ser, en definitiva, como es.

A Alberto, Carlos, Gorka, Guille, Héctor, Javi, Pablo y Sergio, por el día a día en la universidad, por haber dado ejemplo y por su ayuda.

A los compañeros de la escuela a pesar de que les guste el fútbol, al Chico, a Elisota, al Espada, a Fur, a Gabi, a Gélez, a la Gemio, a Ivic, a Jones, a Kano, a Lel, a Mili, al Molo, a Nac, al Niño, a Pay, Sari y al Teju. Y a Sonia y Raquel, como quiera que se llamen en realidad.

De aquí hay que destacar que sin la ayuda especial de Javi y de Pablo, de Manuel y de Dani, las simulaciones no habrían acabado a tiempo.

A Jörg Mühlbacher, que hizo de mi vida en Linz, donde aprendí tanto, uno de los mejores momentos de mi vida, y fue y es un gran amigo.

A mis amigos de Irlanda, Aitor, Corla, Javier y Sonia, aunque nos estemos yendo.

A la gente de Berlín de Siemens, sobre todo a Mauricio y Anna, y a los erasmus de Linz; Nadia, por mostrarme Roma, y a Serena, Franzi y Franceschina. Y a Gunther, mi compañero de habitación.

Índice general

1. Introducción: Objetivos y organización del Proyecto **1**
1.1. Objetivos 2
1.2. Organización del Proyecto 3

I Presentación teórica 5

2. Problema de Steiner **7**
2.1. Introducción 7
2.1.1. Problema de Steiner en un grafo 8
2.2. Complejidad del problema de Steiner y casos particulares . . . 11

3. Algoritmos Genéticos Híbridos **12**
3.1. Introducción 12
3.2. Los algoritmos genéticos en el contexto de la inteligencia artificial 13
3.3. Terminología 15
3.3.1. Población, individuo y generación 15
3.3.2. Genotipo, cromosoma, genoma y representación 16
3.3.3. Codificación *ad hoc* 16
3.3.4. Fenotipo 17
3.3.5. Evaluación, objetivo y adaptación 18
3.3.6. Criterios de parada 18
3.3.7. Selección 19
3.3.8. Cruce o recombinación 20
3.3.9. Mutación 21
3.3.10. Inversión 22
3.3.11. Transformación lineal 22
3.4. Algoritmos Híbridos 24

3.5. Teorías matemáticas del funcionamiento de los AG 24
3.5.1. Teorema de los esquemas 25
3.5.2. HBC (Hipótesis de los bloques de construcción) 29
3.6. Dificultad en los AG 30
3.6.1. Clases de dificultad 31
3.6.2. Disrupción por cruce (epistasis) 32
3.6.3. Deriva genética 33
3.6.4. Dificultad desde el punto de vista del espectro 33
3.6.5. Engaños 35

4. Diseño del algoritmo **37**
4.1. Decisiones de diseño 37
4.1.1. Representación 37
4.2. Descripción del HRD. 42
4.2.1. Cálculo del grafo de distancias $D(G)$ 43
4.2.2. Construcción del subgrafo G_1 de $D(G)$ inducido por W 43
4.2.3. Cálculo de un árbol generador mínimo T_1 de G_1 43
4.2.4. Expansión de T_1 en G_2 44
4.2.5. Cálculo de un árbol generador mínimo T_2 de G_2 44
4.2.6. Poda de las ramas sobrantes 44
4.2.7. Diferencias de diseño con Esbensen [14] 44
4.3. Diseño de la transformación lineal. 47
4.3.1. Análisis de esquemas para confeccionar una transformación 48
4.3.2. Matriz directa del análisis de perturbación mínima de la matriz identidad 50
4.3.3. Matriz directa del análisis de permutación de la identidad 52
4.3.4. Matriz de una permutación del esquema con perturbación mínima de la identidad 54
4.3.5. Generación del esquema a partir de las estadísticas de la población 55

5. Software **58**
5.1. El programa `gadnh` 58
5.1.1. Requisitos no funcionales 58
5.1.2. Requisitos funcionales 59
5.1.3. Ejemplo de un fichero de resultados 65
5.2. Clases 68
5.2.1. eoSchemaStat $<$ EOT $>$ 69
5.2.2. eoEvalFuncWithGraph $<$ EOT $>$ 73
5.3. Implementación del HRD 74

5.3.1. Cálculo del grafo de distancias $D(G)$ 74
5.3.2. Construcción del subgrafo G_1 de $D(G)$ inducido por W 75
5.3.3. Cálculo de un árbol generador mínimo T_1 de G_1 75
5.3.4. Expansión de T_1 en G_2 75
5.3.5. Cálculo de un árbol generador mínimo T_2 de G_2 75
5.3.6. Poda de las ramas sobrantes 75
5.4. Árbol de desarrollo 75
5.4.1. `dnh.cpp` e `include/dnh.hpp` 77
5.4.2. `include/eoEvalFuncWithGraph.hpp` 77
5.4.3. `include/eoSchemaStat.hpp` 77
5.4.4. `include/gadnh.hpp` 77
5.4.5. `include/graphComplete_Prune.hpp` 78
5.4.6. `include/rccstd.hpp` 78
5.4.7. `include/steiner.hpp` 79
5.4.8. `steiner_io.cpp` e `include/steiner_io.hpp` 79
5.4.9. `Makefile` 80
5.4.10. `m_dnh.cpp` e `m_steiner_io.cpp` 80
5.4.11. `m_gadnh.cpp` 80
5.5. Guiones de `bash` 80
5.5.1. `backup.sh` 80
5.5.2. `gadnh.sh` 81
5.5.3. `results/analyze.sh` y `metaanalyze.sh` 81
5.5.4. `results/domeans.sh` y `plotresults.sh` 82
5.6. La biblioteca PARI 82

II Resultados de laboratorio **85**

6. Resultados experimentales **87**
6.1. Descripción de las simulaciones 87
6.1.1. Problemas sobre los que se trabaja 87
6.1.2. Limitación por tiempos de ejecución 88
6.1.3. Limitación por tiempo de convergencia 89
6.1.4. Selección de problemas y parámetros del análisis 92
6.2. Complejidad temporal del algoritmo 93
6.3. Efecto en el tiempo de convergencia de la TLU 93
6.3.1. Caso `E=0 S=1` 96
6.3.2. Caso `E=1 S=1` 97
6.3.3. Caso `E=1 S=0` 97
6.3.4. Caso `E=0 S=0` 97
6.3.5. Resumen de resultados experimentales de la primera fase 97

6.4. Extensión del estudio a otros problemas 102
6.4.1. Caso `E=1 S=0` 102

7. Conclusiones y trabajo futuro **107**
7.1. Conclusiones 107
7.1.1. Sobre la Transformación lineal 107
7.1.2. Sobre el desarrollo de un entorno de trabajo 108
7.2. Trabajo futuro 108

III Anexos **111**

A. Presupuesto **113**
A.1. Documentación 113
A.1.1. AGH (Algoritmos genéticos híbridos) 113
A.1.2. HRD (Heurístico de Red de Distancia) 114
A.1.3. Biblioteca BGL (Boost Graph Library) 114
A.1.4. Biblioteca EO (Evolutionary Objects) 115
A.1.5. Herramientas de representación gráfica 115
A.1.6. Costes no incluídos 115
A.2. Software 116
A.2.1. Diseño 116
A.2.2. Banco de pruebas. Biblioteca OR (Operations Research).116
A.2.3. HRD (Heurístico de Red de Distancias) 116
A.2.4. AGH (Algoritmo Genético Híbrido) 117
A.2.5. Guiones de análisis de resultados 117
A.3. Pruebas de laboratorio 118
A.3.1. Preparación de las simulaciones 118
A.3.2. Simulaciones 118
A.4. Documentación del Proyecto 118
A.4.1. Documentación del software 118
A.4.2. Memoria del Proyecto 119
A.5. Material y Equipos 119
A.6. Coste y duración total del Proyecto 121

B. Introducción a la teoría de la complejidad **123**

C. Estudio previo de matrices unitarias **126**

D. Glosario de términos
español castellano – inglés **130**

Índice de figuras

2.1. Grafo con ciclo de longitud negativa. 11

3.1. Diagrama de bloques de un algoritmo genético. 13
3.2. Clasificación de los algoritmos genéticos en la IA 14
3.3. Espacios de evolución, búsqueda y evaluación. 17
3.4. Ejemplo de cruce para el modelo canónico. 21
3.5. Espacio transformado. 23

4.1. Grafo de Esbensen. 38
4.2. Codificación para el ejemplo de Esbensen. 40
4.3. Grafo G_1 de $D(G)$ inducido por W. 45
4.4. Árbol generador mínimo T_1 45
4.5. Grafo expandido G_2. 46
4.6. Árbol generador mínimo T_2. 46
4.7. Árbol podado. 47
4.8. Probabilidad de que T sea invertible 53
4.9. Transformaciones T y P 56

5.1. Jerarquía de la clase eoSchemaStat $<$ EOT $>$ 69
5.2. Jerarquía de la clase eoEvalFuncWithGraph $<$ EOT $>$ 73
5.3. Árbol de desarrollo del software del Proyecto. 76

6.1. Complejidad, número de vértices. 93
6.2. Complejidad, número de vértices seleccionados. 94
6.3. Complejidad, número de aristas. 94
6.4. Complejidad, longitud del genotipo. 95
6.5. Complejidad, $\mathcal{O}(mn^2)$. 95
6.6. Complejidad, $\mathcal{O}(m^2)$. 96
6.7. Resultados de convergencia para `steinb10.txt` con `E=0 S=1`. 98
6.8. Resultados de convergencia para `steinb11.txt` con `E=0 S=1`. 98

6.9. Resultados de convergencia para `steinb10.txt` con `E=1 S=1`. 99
6.10. Resultados de convergencia para `steinb11.txt` con `E=1 S=1`. 99
6.11. Resultados de convergencia para `steinb10.txt` con `E=1 S=0`. 100
6.12. Resultados de convergencia para `steinb11.txt` con `E=1 S=0`. 100
6.13. Resultados de convergencia para `steinb10.txt` con `E=0 S=0`. 101
6.14. Resultados de convergencia para `steinb11.txt` con `E=0 S=0`. 101
6.15. Resultados de convergencia para `steinb02.txt` con `E=1 S=0`. 103
6.16. Resultados de convergencia para `steinb04.txt` con `E=1 S=0`. 103
6.17. Resultados de convergencia para `steinb06.txt` con `E=1 S=0`. 104
6.18. Resultados de convergencia para `steinb07.txt` con `E=1 S=0`. 104
6.19. Resultados de convergencia para `steinb09.txt` con `E=1 S=0`. 105
6.20. Resultados de convergencia para `steinb14.txt` con `E=1 S=0`. 105
6.21. Resultados de convergencia para `steinb15.txt` con `E=1 S=0`. 106

A.1. Reparto de costes del Proyecto. 122
A.2. Duración relativa de las tareas. 122

Índice de cuadros

4.1. Ejemplo comentado de un fichero con los datos de un problema. 39

6.1. Estimación de los tiempos de ejecución. 88
6.2. Estimación de los tiempos de convergencia sin la TL. 90
6.3. Valor del árbol de Steiner mínimo y resultado del HRD. 91

A.1. Resumen de costes asociados a tareas. 120
A.2. Resumen de costes de material y equipos. 121
A.3. Coste total del Proyecto. 121

Capítulo 1

Introducción: Objetivos y organización del Proyecto

Los algoritmos genéticos híbridos han demostrado ser métodos eficaces para resolver problemas complejos. Esbensen presentó en 1995 [14] un algoritmo genético hibridado con el Heurístico de Red de Distancias que conseguía buenos resultados sobre los problemas de la biblioteca OR para el Problema de Steiner en un grafo.

Por otro lado, Liepins y Vose [36] han demostrado que, para cualquier problema con una función de evaluación monomodal, existe también una representación para la que el problema resulta sencillo para un algoritmo genético, y que existe una transformación para cada representación que puede proyectarla en dicho espacio donde el algoritmo genético funciona bien.

La descomposición de las funciones de evaluación mediante un análisis de Walsh es una herramienta muy utilizada entre los investigadores del campo de los algoritmos genéticos para intentar predecir sus prestaciones. Aunque se ha demostrado [19, 43] que a menos que toda la energía esté en coeficientes de orden 1, no hay una base teórica que permita relacionar el espectro de la función de evaluación con las prestaciones del algoritmo genético, trabajos recientes como el de Salcedo [45] sí que indican la prometedora línea de trabajo que abre la búsqueda de transformaciones para comprimir el espectro, aunque aún es necesario un estudio exhaustivo sobre problemas reales.

En este Proyecto proponemos una transformación lineal unitaria generada a partir de las estadísticas de los genotipos, teniendo en cuenta los trabajos citados anteriormente.

Se ha tenido en cuenta el hecho de que trabajar con transformaciones lineales supone una aproximación de primer order a la resolución del problema, ya que la epistasis consiste en una relación no lineal entre los genes, y se han demostrado [45] las limitaciones de las transformaciones lineales unitarias para comprimir el espectro, pero no tenemos constancia de que se haya desarrollado ninguna teoría para transformaciones no lineales en este campo, y el desarrollo de una teoría de este tipo sobrepasaría ampliamente el ámbito de este Proyecto.

Muchas veces el investigador se ve limitado en sus trabajos porque no dispone de las herramientas adecuadas. Éstas pueden no existir, o ser demasiado específicas para un campo y no permitir su interacción con otras, o demasiado generales y poco eficientes. Un problema especialmente grave son las herramientas propietarias, que impiden crear un fondo de conocimiento común.

Todos los programas y bibliotecas de este Proyecto, así como las plataformas de trabajo, se han seleccionado a partir de estas consideraciones, de forma que permitieran al equipo al cargo del mismo tener el mayor control posible sin depender de agentes externos, aprovechar la base de conocimiento ya creada por la comunidad científica en cada campo de interés, y permitir continuar el desarrollo a partir del punto final del trabajo realizado.

1.1. Objetivos

Los objetivos principales de este Proyecto son:

Transformación lineal Creación y estudio del efecto de una transformación lineal sobre las prestaciones de un algoritmo genético híbrido.

Desarrollo de un entorno de trabajo Selección de uno o varios lenguajes de programación y búsqueda de bibliotecas especializadas en cada uno de los campos que cubre el Proyecto.

A partir de las decisiones tomadas para cumplir los objetivos principales, se definieron otros objetivos secundarios a lo largo del tiempo de vida del Proyecto:

C++ Aprendizaje de este lenguaje, que tiene una difusión muy amplia entre la comunidad científica, como base para el desarrollo de la mayor parte del software.

Guiones de `bash` Aprendizaje del lenguaje para crear pequeños programas de análisis y visualización de resultados.

Biblioteca EO Dominio de esta biblioteca libre, de C++, especializada en Objetos Evolutivos.

Biblioteca BGL Dominio de esta biblioteca libre, de C++, especializada en estructuras de datos y algoritmos de grafos.

Biblioteca PARI Dominio de esta biblioteca libre, de C++, especializada en teoría de números.

LaTeX Como entorno libre para la edición de la documentación del Proyecto.

`octave` Uso de este clon libre del conocido programa Matlab® de MathWorks©, que está integrado con `gnuplot` para la representación de gráficas, y permite la ejecución desde un guión.

`gnuplot` Uso de esta herramienta libre para la representación de gráficas.

`dia` Uso de esta herramienta libre para la representación de diagramas.

1.2. Organización del Proyecto

La memoria se encuentra estructurada en tres partes. La primera corresponde a la descripción del problema y exposición teórica del estudio. Consta de cuatro capítulos:

Problema de Steiner Breve historia del problema de Steiner, y su particularización sobre la métrica de mínima distancia en un grafo. Además se expone su complejidad y algunos casos particulares.

Algoritmos genéticos híbridos Introducción a la teoría de los algoritmos genéticos y su contexto en el campo de la Inteligencia Artificial, el uso de algoritmos locales para crear algoritmos híbridos, las variaciones sobre el algoritmo canónico, las teorías matemáticas que los sustentan y las dificultades con las que tienen que bregar.

Diseño del algoritmo Descripción desde un punto de vista funcional del algoritmo de Esbensen, las variaciones introducidas en este Proyecto, el diseño de la transformación lineal y su relación con los esquemas.

Software Descripción desde un punto de vista de implementación del algoritmo, que se presenta en forma del programa `gadnh`, con los requisitos funcionales y no funcionales, sus componentes, el árbol de desarrollo, y algunos programas auxiliares dirigidos especialmente a la extracción y presentación de resultados.

Los trabajos de laboratorio y las conclusiones pertenecen a la segunda parte, que consta de dos capítulos.

Resultados experimentales Se presentan los análisis previos y las decisiones tomadas a partir de ellos para la fase de simulaciones, así como los resultados obtenidos.

Conclusiones y trabajo futuro Se resumen las conclusiones a las que condujo el Proyecto, su relación con los objetivos iniciales y las líneas de trabajo futuro que quedan abiertas.

Por último, la tercera parte se compone de cuatro capítulos en forma de anexos.

Presupuesto Recoge los costes y duración de las tareas y equipos necesarios para la realización del Proyecto.

Introducción a la teoría de la complejidad Para el lector curioso.

Estudio previo de matrices unitarias Donde se describe el desarrollo de una de las matrices con las que se trabajó en las fases previas del Proyecto.

Glosario de términos español castellano - inglés Con algunos términos que se suelen usar en inglés y que en el Proyecto se han traducido a nuestro idioma.

Parte I

Presentación teórica

Capítulo 2

Problema de Steiner

2.1. Introducción

El problema de Steiner propone la busca de un árbol de peso mínimo que conecte una serie de puntos en un espacio métrico (S, d). S es el espacio de los m puntos a unir y los puntos auxiliares (llamados **puntos o vértices de Steiner**), y d es una distancia $d : S^2 \mapsto \mathbb{R}_0^+$. Las condiciones que tienen que cumplir los puntos en un espacio métrico, $x, y, z \in S$, son tres[1]:

$$d(x,y) \geq 0, \text{ y } d(x,y) = 0 \Leftrightarrow x = y \tag{2.1a}$$

$$d(x,y) = d(y,x) \tag{2.1b}$$

$$d(x,y) \leq d(x,y) + d(y,z) \tag{2.1c}$$

Los antecedentes del problema se hayan en el s. XVII, cuando P. Fermat se preguntó cómo encontrar un punto en un triángulo de forma que la suma de las distancias de este punto a los vértices sea lo más pequeña posible. La solución la dio E. Torricelli, y el punto se conoce con el nombre de *punto de Torricelli* o *primer punto de Fermat*. J. Steiner fue un matemático del s. XIX, pero no está clara cuál fue su contribución a este problema. El Problema de Steiner es una generalización del problema de Fermat para unir m puntos, con r puntos auxiliares (en el problema de Fermat, $m = 3,\ r = 1$, con métrica euclídea).

Al menos en las métricas más utilizadas (euclídea, rectilínea, distancia de Hamming, distancia del camino más corto en un grafo) el problema es PN-Completo [16]. Tiene muchas aplicaciones, por lo que se ha investigado

[1] La Ec. (2.1c) es la desigualdad triangular.

profusamente [26], por ejemplo, en problemas de diseño VLSI para la métrica rectilínea, problemas de centrales de conmutación para la métrica de orientación uniforme (una generalización de las métricas euclídea y rectilínea), o problemas de encaminamiento de paquetes multidifusión para la métrica de camino más corto en un grafo.

La primera aproximación para resolverlo es buscar el árbol generador mínimo que une los m puntos. En el caso euclídeo, por ejemplo, se busca un árbol generador sin puntos adicionales; en el caso de distancia en un grafo, se busca un árbol generador para todo el grafo, y se podan las ramas que no terminan en los vértices seleccionados. Esto se conoce como el Problema del Árbol Generador Mínimo, y se puede resolver en tiempo polinomial.

Para la métrica del camino más corto en un grafo, que es de la que trata este Proyecto, la mejora máxima que puede proporcionar un árbol de Steiner mínimo sobre un árbol generador mínimo es de sólo un 14 % aproximadamente [17, 11, 12], por lo que hay que considerar en qué casos vale realmente la pena resolver un problema con dificultad PN, en lugar del otro, que por ejemplo con el algoritmo de Prim tiene complejidad $\mathcal{O}(|V|^2)$. Un caso claro son las instalaciones telefónicas, donde el alto coste de la obra compensa toda la computación adicional.

La tasa de prestaciones (TP) de un heurístico aplicado al problema de Steiner se define como la cota superior de la razón entre el peso del árbol conseguido por éste y el óptimo (el peso del árbol de Steiner mínimo). Para cualquier métrica, un árbol generador mínimo consigue $TP = 2$ [50]. Los mejores heurísticos para la métrica del camino más corto en un grafo consiguen del orden de $TP = 1{,}64$, $TP = 1{,}55$ en tiempo polinomial [32, 55, 40].

2.1.1. Problema de Steiner en un grafo

Este caso particular del problema de Steiner se conoce también como Problema de la red de Steiner –una red es una dupla (G, d) donde G es el grafo y d una distancia definida sobre las aristas–, y corresponde al problema definido sobre la métrica del camino más corto en un grafo no dirigido. Los términos distancia, peso y coste se intercambian en el uso (aunque se suele hablar de coste de una arista, distancia entre dos vértices, y peso de un árbol), y pueden dar una medida del retardo, distancia física, ancho de banda o coste de un enlace.

Damos algunas definiciones de términos relacionados con grafos y con el problema de Steiner que se emplean frecuentemente en este Proyecto, siguiendo la misma terminología que [14], que a su vez la toma de [2].

Un grafo no dirigido G es una dupla $G = (V, E)$ con una función de coste, $c : E \mapsto \mathbb{R}_0^+$, asociada a E, donde $V = \{v_0, v_1, \dots, v_{n-1}\}$ es el conjunto de los vértices, y $E = \{e_0, e_1, \ldots, e_{k-1}\}$ el de las aristas.

En el problema de Steiner, V se compone de dos conjuntos disjuntos, $V = W \cup S$. W son los vértices a unir, o vértices seleccionados, y R es el conjunto de vértices auxiliares. Los vértices auxiliares escogidos para una solución concreta del problema se llaman *vértices de Steiner*, $S \subseteq R$. Las dimensiones de cada conjunto se denominan

$$\begin{aligned} n &= |V| \\ m &= |W|,\ 0 \leq m \leq n \\ r &= n - m \\ s &= |S|,\ 0 \leq s \leq r \end{aligned}$$

Definición 2.1 (Distancia entre dos vértices) *Dado un grafo $G = (V, E)$, la* distancia $d(a, b)$ entre dos vértices $a, b \in V$ *se define como el paseo de coste mínimo entre ellos.*

Definición 2.2 (Coste de un grafo) *Se llama* coste de un grafo $G = (V, E)$ *a la función*

$$c(G) = \sum_{i=0}^{m-1} c(e_i)$$

donde

$$E = \{e_0, e_1, \ldots, e_{m-1}\}$$

Definición 2.3 (Árbol de Steiner) *Se llama* árbol de Steiner para W en G *a cualquier subgrafo acíclico G' de G tal que $W \subseteq V'$.*

Definición 2.4 (Árbol mínimo de Steiner) *Se llama* árbol mínimo de Steiner para W en G *a un árbol de Steiner con coste mínimo.*

Definición 2.5 (Problema de Steiner en un grafo) *Sea G un grafo conectado no dirigido, $G = (V, E)$, con una función de coste positiva, $c : E \mapsto \mathbb{R}_0^+$, y sea W un subconjunto de vértices, $W \subseteq V$. El* problema de Steiner en un grafo *se define como el cálculo de un subgrafo conectado de G, $G' = (V', E')$, tal que $W \subseteq V'$ y $c(G')$ sea mínimo.*

Esta definición no está restringida ni a grafos planares, ni completos, ni a que c cumpla la desigualdad triangular [14]. En cuanto a los grafos no completos, el lema siguiente [27] muestra que hay una equivalencia con el grafo completado $D(G)$, que es el grafo de distancias de G.

Lema 2.6 *Sea $G = (V, E)$ un grafo, y sea su conjunto de vértices la unión disjunta $V = W \cup R$, donde W es el conjunto de vértices mínimo de cualquier árbol de Steiner. Sea c una función $c : E \mapsto \mathbb{R}_0^+$, y d la función de distancia en la red (G, c). Entonces el coste de un árbol de Steiner mínimo para la red $(D(G), d)$ es el mismo que el coste de un árbol de Steiner mínimo para la red original (G, c).*

El lema anterior permite pasar de un caso no métrico al caso métrico, ya que la función de costes d de las aristas de $D(G)$ siempre cumple la desigualdad triangular, con independencia de la función de coste c de las aristas de G [35, 14]. Esto nos permite aplicar el lema siguiente [17], que acota el número máximo de vértices de Steiner necesario.

Lema 2.7 (Cota superior del número de vértices de Steiner) *Sea $G = (V, E)$ un grafo completo, y sea su conjunto de vértices la unión disjunta $V = W \cup R$, donde W es el conjunto de vértices mínimo de cualquier árbol de Steiner. Sea c una función $c : E \mapsto \mathbb{R}_0^+$ que satisface la desigualdad triangular* (2.1c). *Entonces existe un árbol de Steiner mínimo para la red (G, c) con a lo sumo $m - 2$ vértices de Steiner.*

La condición de costes positivos $c : E \mapsto \mathbb{R}_0^+$ es importante. Cuando hay costes negativos, pueden existir ciclos de coste negativo, lo que permite hallar paseos de coste arbitrariamente pequeño entre dos vértices[2], como queda ilustrado en la Fig. 2.1. Podemos evitar estos casos hablando de rastros en lugar de paseos, pero en cualquier caso en nuestras instancias siempre vamos a encontrar costes positivos para las aristas, por lo que no tendremos este problema.

[2]Una solución errónea que suele venir a la mente consiste en sumar una constante C a los costes de todas las aristas, de forma que sean todos positivos. Esto no es posible porque para cada camino $(e_0, e_1, \ldots, e_{k-1})$ el coste se incrementa una cantidad $k \cdot C$, por lo que en general $D((G, c+C)) \neq P \cdot D((G, c))$ para una constante P, así que las distancias relativas entre los vértices cambian.

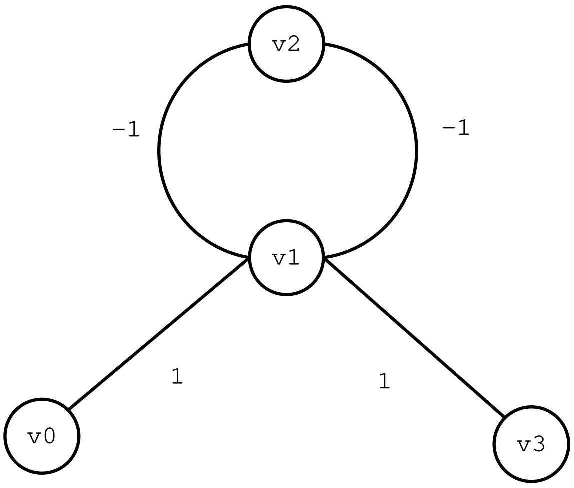


Figura 2.1: Grafo con ciclo de longitud negativa.

2.2. Complejidad del problema de Steiner y casos particulares

El caso general del problema de Steiner tiene dificultad PN [31], como adelantamos en la sección anterior para varias métricas. El algoritmo de Lawler [35], por ejemplo, tiene complejidad polinómica en $r = n - m$, y exponencial en m.

Si $m = 2$, el problema se conoce como *problema del camino más corto*, que consiste en hallar uno de los caminos de distancia mínima entre dos vértices. Este problema se puede resolver con el algoritmo de Dijkstra (válido para grafos dirigidos y no dirigidos), cuya versión modificada con cola de prioridad tiene complejidad $\mathcal{O}(|E| \log n)$ [27]. Si la cola de prioridad es una pila de Fibonacci, la complejidad se puede reducir [15] hasta $\mathcal{O}(|E| + n \log n)$. Si los costes de las aristas son menores que una constante C no muy grande, se puede llegar [3] a una complejidad $\mathcal{O}(|E| + n\sqrt{\log C})$. En caso de tener costes negativos en las aristas, podemos emplear el algoritmo de Bellman-Ford [27], con complejidad $\mathcal{O}(|E|n)$, siempre y cuando no existan ciclos con coste negativo.

Si $r = 0$ (todos los vértices están en W), el problema es el problema del árbol generador mínimo, del que hablamos en la sección anterior, y que es uno de los problemas sencillos más importantes de la algoritmia de grafos. La primera referencia que conocemos es de Boruvka [6, 7], aunque son más conocidos Kruskal [34] y Prim [39]. El de Prim tiene complejidad $\mathcal{O}(n^2)$, mientras que el de Kruskal [2, 8] tiene complejidad $\mathcal{O}(|E| \log |E|)$, lo que es mejor para grafos huecos.

Capítulo 3

Algoritmos Genéticos Híbridos

3.1. Introducción

El modelo de **algoritmo genético** fue introducido por Holland [25], de-Jong [10] y Goldberg [18] y consiste básicamente en una codificación de los individuos de una población como genotipos o cromosomas con forma de cadenas de bits, que en su forma descodificada o fenotipo son posibles soluciones de la instacia de un problema concreto, además de una función de evaluación de los fenotipos, y una serie de operadores (*selección*, *cruce*, *mutación*, ...) inspirados en los procesos de evolución natural, bajo el paradigma de supervivencia del (estadísticamente) más fuerte.

El diagrama de bloques correspondiente, inspirado en el de Schönauer [47], se encuentra representado en la Fig. 3.1. Los algoritmos híbridos son la combinación de un algoritmo genético con un esquema de búsqueda local (el bloque en el diagrama).

Este diagrama no es completo, dado que es posible encontrar alguna variación u operador que no admita representación en él. Por ejemplo, sería posible definir un operador de aborto, de forma que por cada hijo hubiera una evaluación y, en caso de no obtener una solución válida, sustituir el genotipo defectuoso por otro hasta dar con una que sí lo sea; y para representar este proceso necesitaríamos añadir un bloque condicional y retroalimentación al diagrama.

Sin embargo, consideramos que el diagrama de la Fig. 3.1 es lo suficientemente general como para representar correctamente los elementos principales de los AG y, sobre todo, permitirnos explicarlos con claridad, ya que en la literatura hay cierta confusión en algunos casos.

En este Proyecto se trabajará sobre la base del algoritmo híbrido propuesto por Esbensen [14] (aunque en dicha referencia no se le denomina

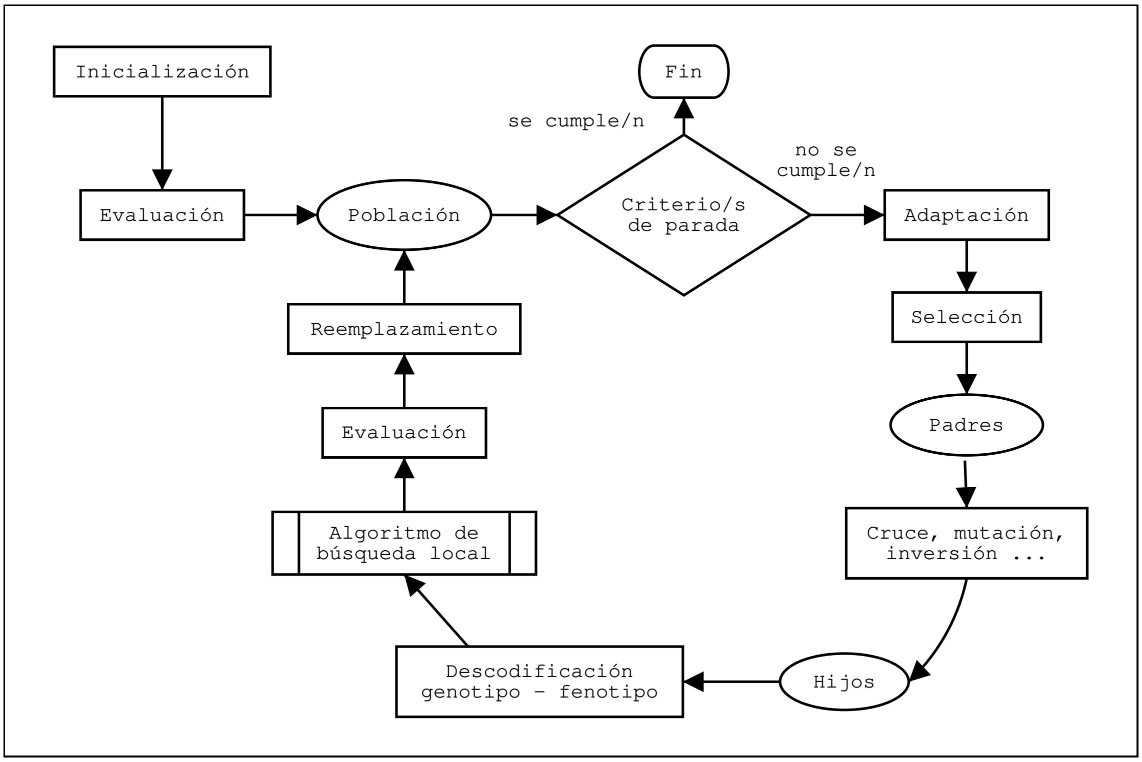


Figura 3.1: Diagrama de bloques de un algoritmo genético.

explícitamente "híbrido"), por lo que dedicaremos una atención especial a la descripción de estos algoritmos de búsqueda global y local.

3.2. Los algoritmos genéticos en el contexto de la inteligencia artificial

Como se puede ver en la Fig. 3.2, los algoritmos genéticos forman parte de los métodos de **computación evolutiva**, unos modelos del campo de la Inteligencia Artificial (IA) que surgieron en los años 60. En ocasiones es difícil diferenciarlos de los otros tipos de algoritmos evolutivos. Normalmente se señala la diferencia en que la codificación del espacio de evolución tiene forma de cadena de bits, por lo que para un genotipo de longitud L el espacio es $\{0,\ 1\}^L$; pero en [13], por ejemplo, se habla de algoritmos genéticos modificados con otras representaciones (lo que aparece con frecuencia en algoritmos híbridos). En general, parece aceptado que el operador principal de los algoritmos genéticos es el de cruce, pero se puede jugar con su peso en la evolución, y según [54] hay varios investigadores que han mostrado que la

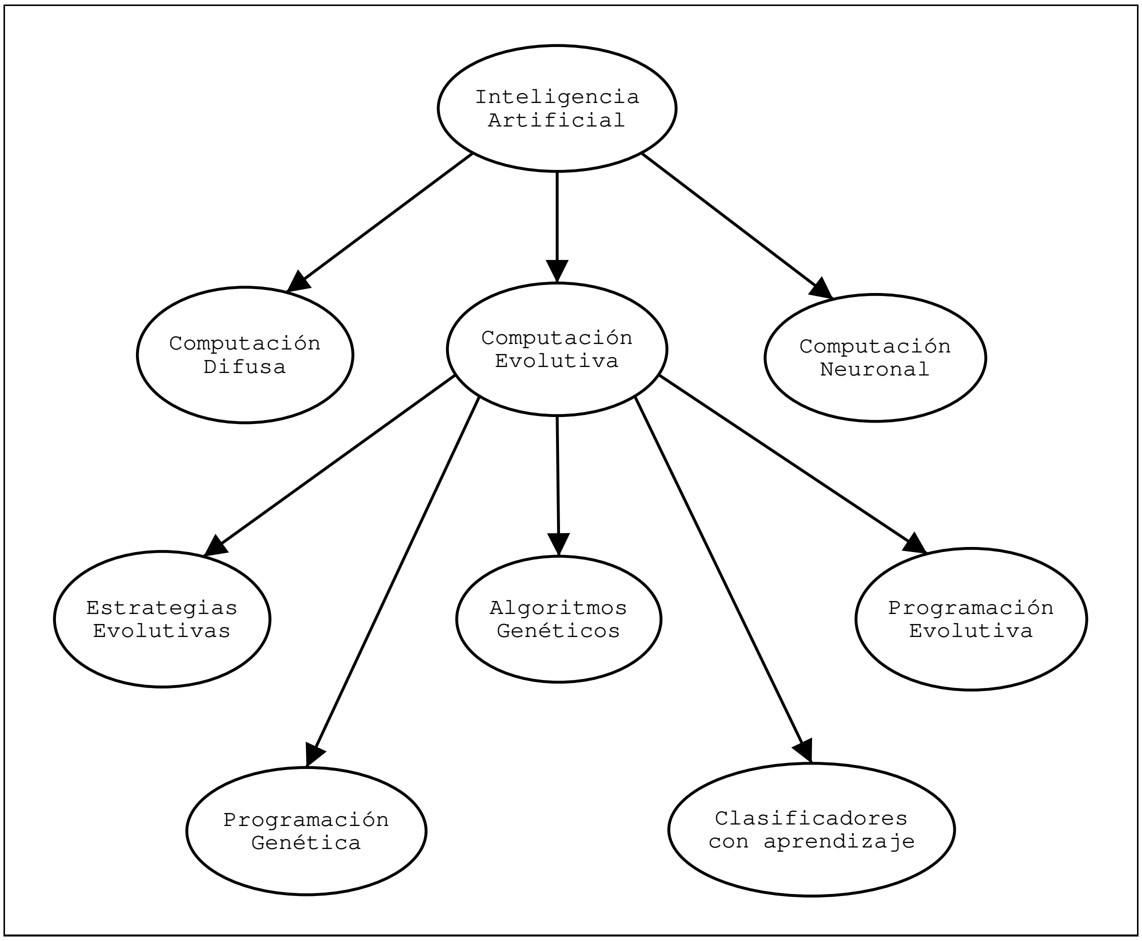


Figura 3.2: Clasificación de los algoritmos genéticos en la IA

búsqueda genética con mutación y sin cruce es robusta con frecuencia.

Hacemos a continuación una enumeración breve, no exhaustiva, de los otros campos de la IA, con sus características principales.

La **programación evolutiva** evoluciona por mutación y selección una población de máquinas de estados finitos, según una medida de lo capaces que son de predecir los símbolos siguientes a partir de unos símbolos de entrada, lo que se emplea como un modelo de la inteligencia.

Las **estrategias evolutivas** codifican los genotipos como cadenas de números reales, y se usan para resolver problemas de optimización con un alto grado de dificultad. Su característica principal es la autoadaptación de los parámetros de la estrategia, como por ejemplo la desviación típica de la amplitud de mutación.

La **programación genética** persigue generar programas que resuelvan un problema concreto de forma automática. Cada individuo es un programa, que tiene un valor de adaptación relacionado con los resultados de la ejecución del mismo. Generalmente se emplean representaciones en forma de árboles, con operadores de cruce específicos.

Los **sistemas clasificadores con aprendizaje** son sistemas dinámicos, basados en reglas de producción, que aprenden por inducción a partir de ejemplos. Las reglas tienen la forma *si <condición>entonces <acción>*, de forma que la acción se lleva a cabo si la condición es cierta. Estos sistemas se puede considerar que son sistemas de producción que emplean los algoritmos evolutivos para generar las reglas.

3.3. Terminología

Tanto en la literatura como en el uso coloquial del lenguage sobre algoritmos genéticos, hay una gran cantidad de términos que se usan de forma imprecisa o incluso incorrecta, como confundir la evaluación con la adaptación, o la probabilidad de mutación con la probabilidad de inversión de bit. En esta sección trataremos de aclarar estas y otras cuestiones relacionadas con el léxico y la semántica, lo que luego nos permitirá exponer la parte científica del Proyecto sin ambigüedades.

Se suele reservar para el modelo de Goldberg [18] la denominación *algoritmo genético canónico*, o *modelo canónico*, y se corresponde con unos operadores y codificación determinados. Al hablar de cada componente, indicaremos cuál es, de entre sus variaciones, la que corresponde al modelo canónico.

3.3.1. Población, individuo y generación

Podemos definir **individuo** como cada uno de los componentes de una **población**. Esto incluye su codificación (*genotipo*) y su expresión en la instancia del problema (*fenotipo*), aunque a veces se usa el término para referirse únicamente al genotipo.

La población de cada bucle del algoritmo recibe el nombre de **generación** (la generación inicial, que es aleatoria, es la generación 0), y existe la posibilidad de que su tamaño sea fijo –como en el modelo canónico– o variable.

3.3.2. Genotipo, cromosoma, genoma y representación

Originalmente se llamó **genotipo** [25] o **cromosoma** [46], a la cadena de bits que codifica los parámetros del problema. Goldberg [18], llama **cadenas** a los cromosomas, y **estructuras** a los genotipos (que son cadenas de bits), según el léxico *artificial* asociado a la computación. A veces incluso se llama **genoma** al genotipo.

En biología no son sinónimos, sin embargo: Toda la información genética del organismo se llama genoma; muchos organismos tienen varios cromosomas en cada célula, lo que se llama *fenómeno de poliploidad*; cada cromosoma a su vez está compuesto por genes, que son cadenas de ADN (Ácido DesoxirriboNucleico), que codifican la síntesis de cada proteína. También hay cadenas de ADN, que separan los genes, pero que no tienen una función de codificación (en realidad, no se sabe muy bien qué función tienen), y se llaman *ADN basura*. El genotipo es el conjunto de los genes de un genoma.

Los valores que puede tomar cada gen se llaman **alelos** (en el caso canónico, $\{0,\ 1\}$), y su posición, ***locus***; el primer término sí se ha adoptado en la literatura (una alternativa es *valor del rasgo*), pero el segundo se suele sustituir por otros más matemáticos, como posición.

Otro término que aparece con frecuencia es **representación**, que es equivalente a los demás que significan codificación. Sin embargo, su uso se suele restringir a la teoría de la representación [36] – y más recientemente [43].

3.3.3. Codificación *ad hoc*

En los algoritmos genéticos la codificación es, como hemos dicho, una cadena de bits[1]. La codificación ***ad hoc*** se refiere a otros algoritmos evolutivos, en los que se busca una representación apropiada al problema en concreto que se esté tratando. De hecho, hay dos corrientes actualmente en el campo de los algoritmos genéticos al respecto. Por un lado, las codificaciones *ad hoc* obtienen mejores resultados en problemas reales, pero gran parte de la teoría para entender el funcionamiento de la computación evolutiva se sigue efectuando sobre codificaciones binarias.

Como caso particular de representación, cuando la transformación genotipo – fenotipo es la identidad, y coinciden los espacios de evolución y búsqueda, hablamos de **representación directa**.

[1] Es evidente que, dada la arquitectura de las máquinas con las que trabajamos, todas las codificaciones se reducen a una forma más o menos elaborada de la binaria. Pero la diferencia de concepto es lo suficientemente clara como para no tener que discutir sobre este aspecto, del mismo modo que cuando hablamos de codificación de vectores reales, tenemos presente que en realidad tratamos con vectores de números de coma flotante

En resumen, aunque la codificación *ad hoc* no es un elemento del campo de los algoritmos genéticos, sí conviene en un proyecto o artículo relacionado con éste explicar por qué se ha decidido usar uno u otro tipo de codificación.

3.3.4. Fenotipo

El **fenotipo** es la expresión o descodificación del genotipo en el contexto de una instancia del problema, de forma que sea un candidato a solución de la misma. Esta diferencia es importante, y permite precisamente que los operadores genéticos trabajen en un espacio independiente del espacio de búsqueda, como se ve en el diagrama de la Fig. 3.3.

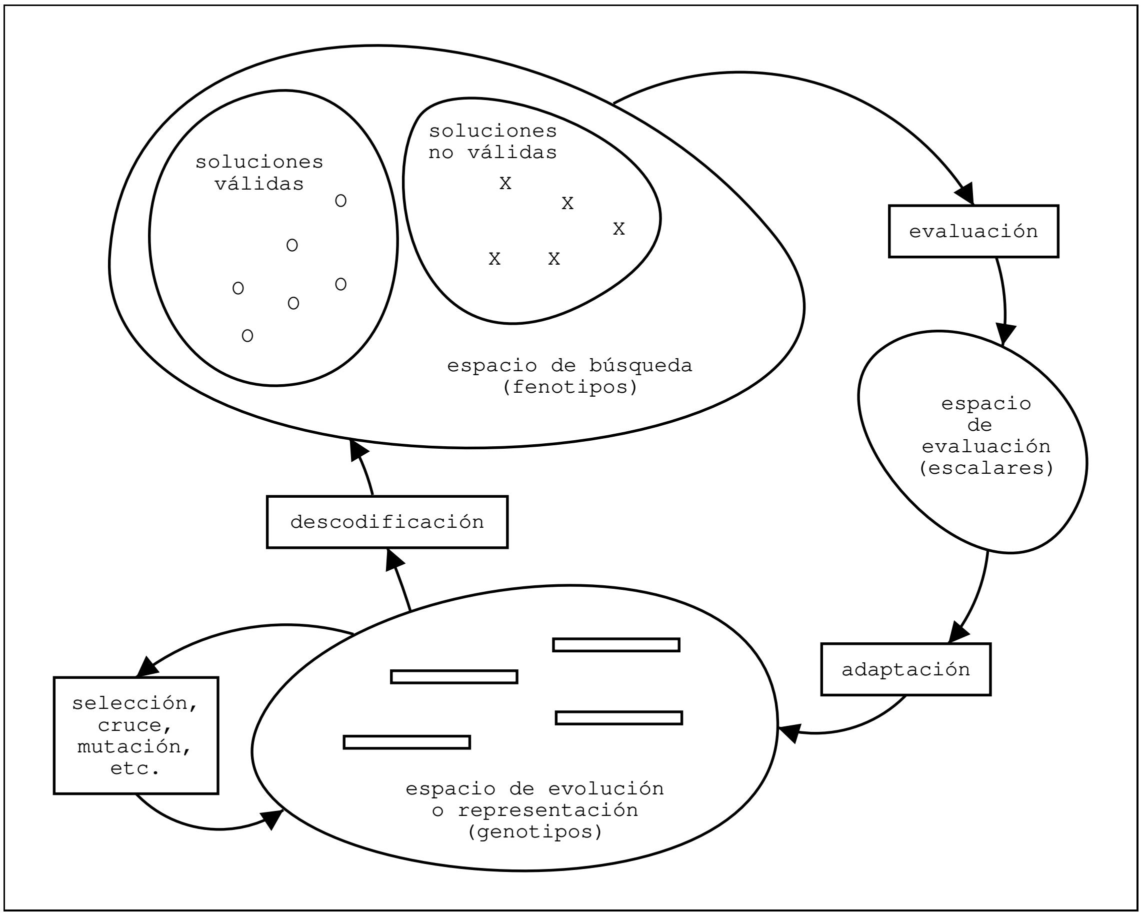


Figura 3.3: Espacios de evolución, búsqueda y evaluación.

3.3.5. Evaluación, objetivo y adaptación

La función de **evaluación**, o función **objetivo**, es una transformación que asigna un valor numérico a cada fenotipo dependiendo de su bondad respecto de un conjunto de parámetros. La función de **adaptación**, en cambio, es una transformación que asigna a cada individuo una medida de sus posibilidades de reproducción.

Ambos términos se intercambian con frecuencia, o se habla sólo de función de adaptación; nosotros creemos que esto no es correcto. El valor de evaluación de un genotipo se calcula con independencia de los otros genotipos, y si es bueno que sea alto o bajo dependerá de cada problema, mientras que el valor de adaptación siempre se define respecto de toda la población, de forma que el valor más alto es el mejor.

Se habla de las distintas funciones que transforman la evaluación en adaptación en el apartado 3.3.7.

3.3.6. Criterios de parada

No hay un límite finito en el número de pasos que puede dar un algoritmo genético para encontrar la mejor solución posible, por lo que hay que fijar unos criterios de parada y una relación entre ellos (que se cumplan todos, al menos uno, ...). Hay muchas posibilidades, de las que enumeramos sólo algunos ejemplos de uso frecuente.

Número máximo de generaciones: El algoritmo corre durante un número N de generaciones, tras el cual se detiene.

Valor de evaluación: El algoritmo corre hasta que se alcanza o supera un valor de evaluación determinado.

Número de generaciones sin mejora: Se guarda un registro de la última generación en la que hubo mejora de un parámetro de evaluación, definido generalmente como el valor del mejor individuo. Se permite que el algoritmo progrese un número N de generaciones sin que haya mejora.

Número mínimo de generaciones: El algoritmo no puede detenerse hasta haber corrido durante un número N de generaciones, tras las cuales generalmente comienza el proceso de contabilizar el número de generaciones sin mejora.

Una medida de control sobre la evolución del algoritmo consiste en el cálculo de la desviación típica de los valores de evaluación en cada generación.

Cuando ésta disminuye significa que se está perdiendo variedad genética y que el algoritmo está convergiendo (no necesariamente a una solución próxima a una óptima).

3.3.7. Selección

Dada una población P en un espacio de evolución S_{evo}, $P = \{p_i\}_{i=0}^{N-1},\ p_i \in S_{\text{evo}}$, y una función de adaptación $f_{\text{ad}} : S_{\text{evo}} \mapsto \mathbb{R}$ (generalmente se normaliza la función de adaptación), la **selección** es una función $f_{\text{sel}} : S_{\text{evo}}^N \times f_{\text{ad}}(S_{\text{evo}}) \mapsto S_{\text{evo}}^N$ que genera otra población (la **población intermedia**) tomando aleatoriamente los individuos de P, de forma que los individuos con mayor valor de adaptación tengan una probabilidad más alta de ser seleccionados. Hay varios métodos para hacer la selección, entre ellos:

Ruleta: Se representa cada individuo por un espacio proporcional a su valor de adaptación, y se seleccionan por *muestreo estocástico con reemplazamiento* haciendo girar la ruleta sucesivamente. Este método es el que se sigue en el modelo canónico.

Muestreo estocástico con resto: Se calcula el valor $f_{\text{ad}}(p_i)/\bar{f}_{\text{ad}}$ para cada individuo (este es el valor de adaptación que se usa en el modelo canónico). Aquellos con un valor mayor que 1,0 añaden a la población intermedia un número de copias igual a su parte entera. La población se completa con una selección aleatoria donde la probabilidad para cada individuo de añadir una copia es igual al resto decimal del valor $f_{\text{ad}}(p_i)/\bar{f}_{\text{ad}}$. Se puede implementar de forma eficiente con el método de *Muestreo Universal Estocástico* [54].

Clasificación: Los individuos se ordenan en función de su evaluación, $p_0,\ \ldots,\ p_{M-1}$, de forma que p_0 es el peor individuo, y p_{M-1} es el mejor. El valor de adaptación de cada individuo se calcula como

$$f_{\text{ad}}(p_i) = \frac{2p-2}{M}\left(\frac{i}{M-1}\right)^e + \frac{2-p}{M}$$

donde $p \in (1,\ 2]$ es la constante de presión selectiva, y e el exponente ($e = 1$ para el caso lineal), de modo que $(2-p)/M \leq f_{\text{ad}}(p_i) \leq p/M$, y en el caso lineal el mejor individuo tiene el doble de posibilidades de ser seleccionado que el individuo medio. La clasificación asegura que la varianza de los valores de adaptación se mantiene constante [53].

Torneo determinista: Se escogen aleatoriamente $N,\ N \geq 2$ individuos, y se selecciona el que tenga el valor de adaptación más alto. Los valores usuales para N están en torno al 5 % - 10 % del tamaño de la población.

Torneo estocástico: Se escogen aleatoriamente parejas de individuos, y se selecciona el que tiene un valor de adaptación más alto con una probabilidad R, $0{,}5 \leq R \leq 1{,}0$.

Cualquiera de estos métodos se puede emplear con o sin elitismo, que consiste en hacer pasar a la generación siguiente al menos una copia del mejor individuo.

3.3.8. Cruce o recombinación

Dada una población P en un espacio de evolución S_{evo}, $P = \{p_i\}_{i=0}^{N-1}$, $p_i \in S_{\text{evo}}$, formado por los individuos seleccionados, el **cruce** es una función $f_{\text{cru}} : S_{\text{evo}}^m \mapsto S_{\text{evo}}^n$ que a partir de la población seleccionada genera otra población (la **población de los vástagos**[2], auque la podemos considerar directamente la **población de los hijos**) tomando de m en m los individuos de P, y cruzando con probabilidad p_c sus materiales genéticos para generar n nuevos individuos. Lo más habitual es tener $m = n$ (para que la población no crezca ni mengüe), en concreto $m = n = 2$, y que el cruce consista en un intercambio de porciones del genotipo a partir de k puntos de cruce elegidos aleatoriamente. Algunos tipos empleados con frecuencia son:

Por un punto: Es el método usado en el modelo canónico, $m = n = 2$, y $k = 1$. En la Fig. 3.4 se ilustra este caso con un ejemplo.

Por dos puntos: Otro valor típico para el número de puntos de cruce es $k = 2$, ya que entonces el operador de cruce actúa como si el genotipo codificado como una cadena uniera sus puntas para formar un anillo [10]. Esto es importante porque así la posición de los bits en los esquemas (hablaremos de esquemas en la sección 3.5) no influye en la probabilidad de que esos bits permanezcan juntos después del cruce.

Por N puntos: Es la generalización de los anteriores, y consiste en formar los vástagos con porciones del genotipo de uno y otro padre, alternativamente, cambiando de padre en cada punto de cruce.

Uniforme: Cada bit del vástago tiene una probabilidad p de provenir del primer padre, y una probabilidad $1 - p$ del segundo, por lo que cada gen se hereda de forma independiente a los otros. Ha sido estudiado por [1, 49]. Es en general más disruptivo que el cruce por uno y dos puntos, e insesgado respecto de la distancia de definición. En [54] hay un buen resumen de sus características y otras referencias.

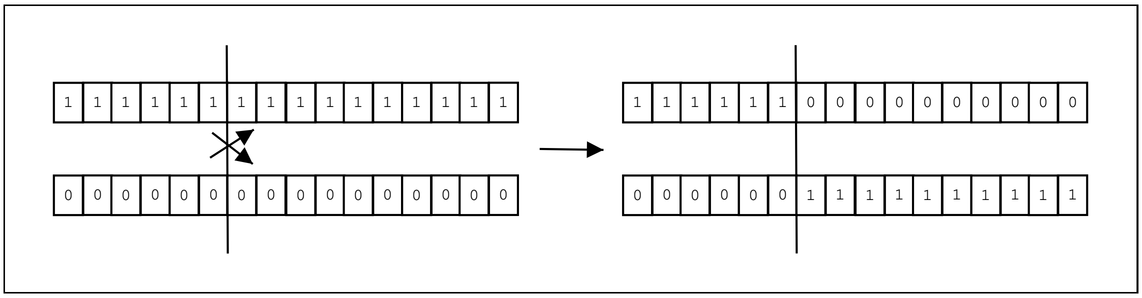


Figura 3.4: Ejemplo de cruce para el modelo canónico.

El cruce tiene una fuerte dependencia con la estructura del espacio de búsqueda y la epistasis, de modo que la representación escogida es esencial para minimizar los efectos de ruptura de esquemas que veremos en el apartadosubsec:agh-dificultad-disrupcion.

3.3.9. Mutación

Dada una población P de hijos (el resultado del cruce) en un espacio de evolución S_{evo}, $P = \{p_i\}_{i=0}^{N-1}$, $p_i \in S_{\text{evo}}$, la **mutación** es una función $f_{\text{mut}} : S_{\text{evo}} \mapsto S_{\text{evo}}$ que introduce una transformación local e independiente de la estructura en el genotipo, consistente en que cada individuo tiene una probabilidad p_{mut}, generalmente muy alta o incluso 1 (no se debe confundir con la probabilidad de que cada bit cambie, que suele ser muy baja), de optar a sufrir un cambio pequeño en los elementos de la cadena, para codificaciones canónicas, o a otro tipo de variaciones para codificaciones *ad hoc*. Típicamente encontramos:

Cambio de bit: Para codificaciones canónicas binarias, Cada bit tiene una probabilidad p_{bit} muy pequeña (típicamente del 1 % o 0,1 %) de cambiar su valor. Este cambio se realiza en unas ocasiones negando el valor del bit que muta, y en otras, dándole un valor aleatorio nuevo, por lo que sólo cambia la mitad de las veces.

Gaussiana: En las codificaciones canónicas con números reales, la mutación se realiza en forma de un ruido blanco con una desviación típica σ pequeña que se añade al valor del gen.

Es evidente que definir las dos probabilidades, p_{mut} y p_{bit}, sobre el algoritmo canónico binario es equivalente a definir una única probabilidad

[2] La población de vástagos es una población intermedia entre la de padres y la de hijos, que es la que se genera entre el cruce y la mutación.

$p_{\text{mut tot}} = p_{\text{mut}} \cdot p_{\text{bit}}$.

La mutación tiene dos funciones muy importantes. La primera es introducir variedad genética para poder alcanzar hiperplanos a los que no se puede llegar sólo mediante el operador de cruce. De esta forma, ningún alelo desaparece completamente de un gen dado.

También es importante, aunque no ha sido lo suficientemente estudiada, la capacidad de la mutación como mecanismo de escalada, por lo que actúa como un algoritmo de búsqueda local. Podemos preguntarnos entonces si cualquier algoritmo genético con mutación no es un algoritmo híbrido; sin embargo, hay una diferencia importante entre la mutación, que realiza una búsqueda local ciega, y los algoritmos locales de los algoritmos híbridos, que tienen información sobre la topología del problema.

3.3.10. Inversión

Dada una población P en un espacio de evolución S_{evo}, $P = \{p_i\}_{i=0}^{N-1},\ p_i \in S_{\text{evo}}$, la **inversión** para una codificación canónica es una función $f_{\text{inv}} : S_{\text{evo}} \mapsto S_{\text{evo}}$ que reordena los genes de la misma. Hay que asegurarse de que la descodificación del genotipo no dependa de la permutación, por lo que se asocia una etiqueta con cada gen

$$(0,\ \pi_0)(0,\ \pi_1)(1,\ \pi_2)(0,\ \pi_3)(1,\ \pi_4)$$

El objetivo consiste en conseguir que los genes que tienen una epistasis fuerte estén próximos para evitar la ruptura de los esquemas en el cruce.

3.3.11. Transformación lineal

Dada una población P en un espacio de evolución S_{evo}, $P = \{p_i\}_{i=0}^{N-1},\ p_i \in S_{\text{evo}}$, la **transformación lineal** para una codificación canónica es una función $f_{\text{TL}} : S_{\text{evo}} \mapsto S_{\text{evo}}$ que se puede representar como $f_{\text{TL}}(p_i) = A \cdot p_i$ donde A es la matriz $A = ((a_i)), a_i \in \{0,\ 1\}$.

Su actuación sobre el diagrama de la Fig. 3.3 se puede ver en la Fig. 3.5. Si la transformación tiene inversa, es arbitrario definir cuál es el sentido de la transformación, y cuál el de la antitransformación (nuestra definición parte de la forma en la que generamos la matriz A en este Proyecto); resulta imprescindible que ésta exista, ya que de este modo se pueden pasar de un espacio a otro los individuos para ser evaluados y para que se realice el cruce, respectivamente.

La inversión es una transformación lineal (en el sentido de la flecha discontinua) que consiste en una permutación de la matriz identidad, por ejemplo,

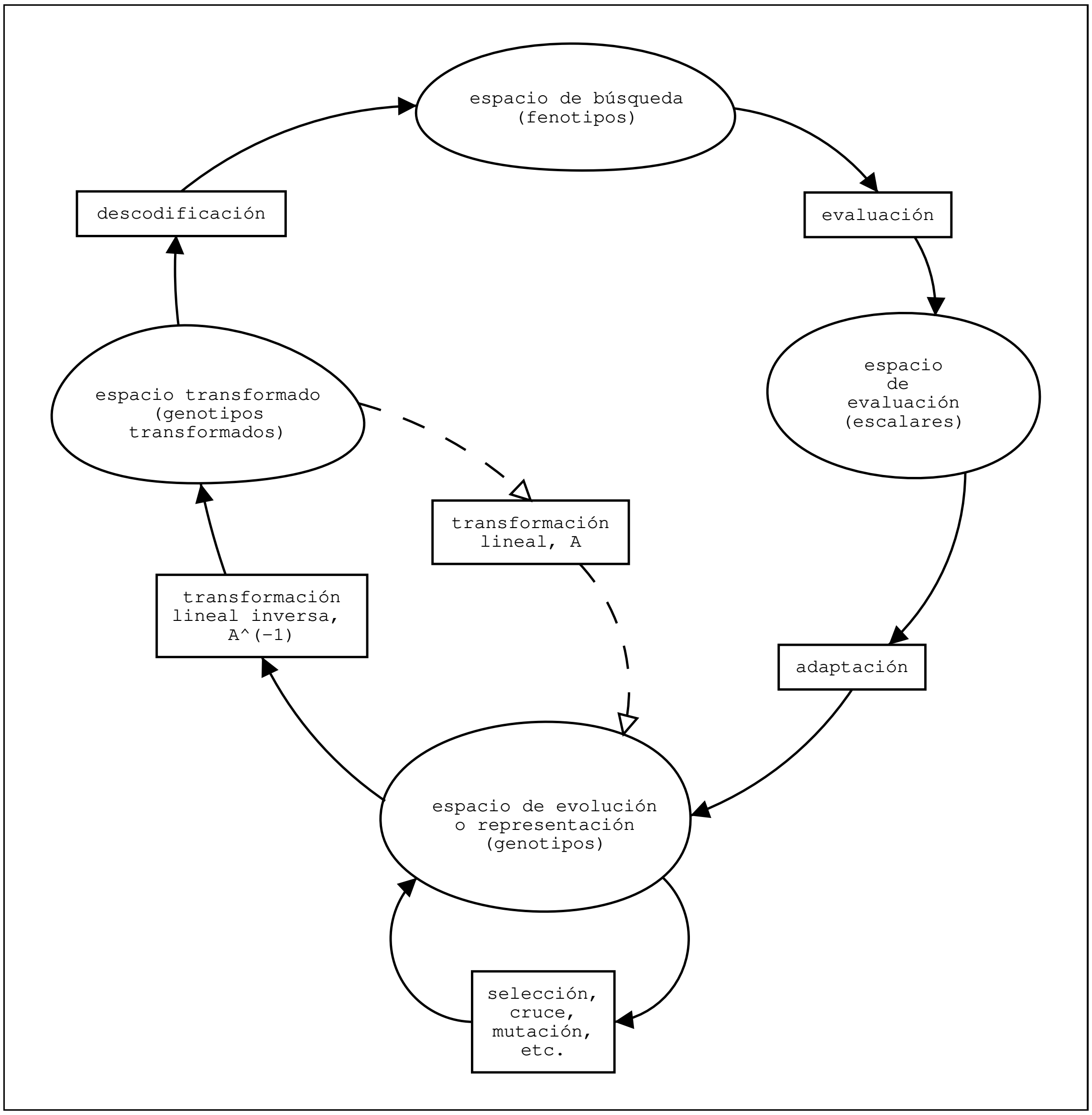


Figura 3.5: Espacio transformado.

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} v_2 \\ v_0 \\ v_1 \end{bmatrix}$$

Hablaremos más sobre transformaciones lineales en la seccionsec:algoritmo-disenoTL.

3.4. Algoritmos Híbridos

Los algoritmos genéticos tienen dificultades para abordar problemas con muchas restricciones, porque es muy complicado encontrar una codificación que evite que el espacio de evolución sea mucho mayor que el de soluciones válidas, y aunque se encuentre, definir unos operadores genéticos que de m individuos válidos obtengan otros tantos individuos que también lo sean.

Uno de los caminos a seguir consiste en definir una codificación *ad hoc* para el problema, buscando una biyección entre los dos espacios; este camino tiene asociados todos los problemas ya conocidos de dificultad del análisis teórico, y de implementación de operadores genéticos específicos, aunque da buenos resultados y se usa en muchos casos prácticos hoy en día [37].

La segunda posibilidad es usar un algoritmo genético y penalizar las soluciones no válidas, como en [28], un trabajo previo al de Esbensen para abordar el problema de Steiner en un grafo. Esto no suele dar buenos resultados [36], porque el algoritmo genético se mueve en un espacio de evolución donde pequeñas variaciones locales pueden dar lugar a variaciones no acotadas en el espacio de búsqueda.

Por último, se puede usar el algoritmo genético, pero en lugar de penalizar las soluciones, se incluye un descodificador especial entre el espacio de evolución y el de búsqueda que cumple dos funciones:

- Imponer las restricciones del problema.
- Aportar información sobre el problema para realizar una búsqueda local.

Esto se conoce como **algoritmo híbrido**, y es el método que se sigue en [14] y en este Proyecto. El "descodificador" al que nos referíamos se conoce formalmente como una **transformación genotipo – fenotipo**, y una explicación de su potencia viene dada en términos de su capacidad para comprimir el espectro en órdenes bajos [29, 30].

3.5. Teorías matemáticas del funcionamiento de los AG

Las teorías matemáticas más conocidas para explicar el funcionamiento de los algoritmos genéticos son el teorema de los esquemas y la hipótesis de los bloques de construcción, aunque se sabe que tienen ciertas limitaciones que les impiden recoger toda su complejidad. Nos restringiremos a la codificación binaria.

3.5.1. Teorema de los esquemas

La teoría parte de Holland [25] y DeJong [10]. Para la exposición seguiremos a Goldberg [18] y a Whitley [54]. Se supone selección proporcional al valor de adaptación con reemplazamiento para el primer padre (después también para el segundo), y cruce por un punto.

Los genotipos son elementos del espacio de evolución, $g_i \in E_{\text{evo}} = \{0,\ 1\}^L$, de longitud L, y un esquema, también conocido como partición de hiperplano o plantilla de similitud, es un subespacio $H_i \subseteq E_{\text{evo}}$ que contiene uno o varios genotipos.

Definición 3.1 (Esquema) *Se llama* esquema *a un vector* $H_i \in \{0,\ 1,\ *\}^L$ *que describe un subespacio del espacio de evolución* $H_i \subset E_{evo}$*, o hiperplano, tal que las componentes* $H_i(j) = 0, 1$ *representan un "0" o un "1", respectivamente, en la posición* j *común a todos los genotipos de* H_i*, mientras que* $H_i(j) = *$ *es un comodín que representa un "0" o un "1" indistintamente.*

Definición 3.2 (Orden de un esquema) *Se llama* orden de un esquema*,* $o(H)$*, al número de "0"s y "1"s del esquema.*

Definición 3.3 (Distancia de definición) *Sean* j *y* k *las posiciones del primer y últimos elementos distintos de "$*$" en un esquema. Se llama* distancia de definición *de un esquema* H*,* $\Delta(H)$*, a la cantidad* $\Delta(H) = k - j$*.*

Goldberg [18] representa la distancia de definición como $\delta(H)$.

Por ejemplo, el esquema $[01 * 1*]$ representa a los genotipos [01010], [01011], [01110] y [01111], y tiene orden 3 y distancia de definición $\Delta(H) = 3 - 0 = 3$. Por convención no se suelen denominar esquemas ni al vector de orden 0, que representa a todo el espacio de evolución (por eso se define $H_i \subset E_{\text{evo}}$ y no $H_i \subseteq E_{\text{evo}}$), ni a los 2^L vectores de orden L, que son los puntos del espacio (los vectores H_i sin elementos "$*$").

El espacio de evolución se puede ver como un hipercubo de dimensión L cuyos puntos son los 2^L vértices, del que cada esquema representa un hiperplano (reservamos el término hipercubo para todo el espacio); hay $3^L - 1$ esquemas distintos, ya que cada elemento de los vectores de esquema puede tener 3 valores ("0", "1" o "$*$"), y no contamos como esquema al de orden 0.

Un ejemplo para $L = 3$ ($3^3 - 1 = 26$ esquemas):

1. $0\ 0\ 0$	4. $0\ 1\ 0$	7. $0 * 0$
2. $0\ 0\ 1$	5. $0\ 1\ 1$	
3. $0\ 0\ *$	6. $0\ 1\ *$	8. $0 * 1$

9. $0 * *$	15. $1\ 1 *$	21. $* 0 *$
10. $1\ 0\ 0$	16. $1 * 0$	22. $* 1\ 0$
11. $1\ 0\ 1$	17. $1 * 1$	23. $* 1\ 1$
12. $1\ 0 *$	18. $1 * *$	24. $* 1 *$
13. $1\ 1\ 0$	19. $* 0\ 0$	25. $* * 0$
14. $1\ 1\ 1$	20. $* 0\ 1$	26. $* * 1$

Cada genotipo se encuentra representado en $2^L - 1 = 2^3 - 1 = 7$ esquemas. Así, al [011] le corresponden los esquemas –entre paréntesis el orden del esquema– [0 1 1] (3), [0 1 $*$] (2), [0 $*$ 1] (2), [0 $**$] (1), [$*$ 1 1] (2), [$*$ 1 $*$] (1) y [$**$ 1] (1). El hecho de que cada individuo muestree simultáneamente tantos hiperplanos, y que cada población esté compuesta por varios individuos, define la propiedad de **paralelismo implícito** [25] de los algoritmos genéticos. El paralelismo implícito se basa en que el aumento de la población en un hiperplano es proporcional al valor de adaptación medio del hiperplano, lo que se conoce como **tasa de muestreo potencial** del hiperplano H.

Podemos formalizar el concepto de la tasa de muestreo potencial si definimos $M(H,\ t)$ como el número de genotipos que muestrean el hiperplano H en la población de la generación t. Sea entonces t_{padres} el instante entre t y las operaciones de evolución, y $\bar{f}$ el valor de adaptación medio de la población; si la reproducción depende sólo de la selección,

$$M(H,\ t_{\text{padres}}) = M(H,\ t) \cdot \frac{f(H,\ t)}{\bar{f}} \tag{3.1}$$

Si llamamos c a la fracción del **valor de adaptación en exceso**

$$\begin{aligned} M(H,\ t_{\text{padres}}) &= M(H,\ t) \cdot \frac{\bar{f} + c\bar{f}}{\bar{f}} \\ &= M(H,\ t) \cdot (1 + c) \end{aligned} \tag{3.2}$$

llegamos a la expresión

$$M(H,\ t) = M(H,\ 0) \cdot (1 + c)^t \tag{3.3}$$

que es una progresión geométrica. Esto quiere decir que el efecto aislado de la selección proporcionará a los esquemas *que aparecieron ya en la inicialización aleatoria* un incremento o decremento de su presencia según su tasa de adaptación en exceso.

Para poder explorar la parte del espacio no muestreada en la inicialización, el algoritmo genético emplea los operadores de cruce y mutación. Estos operadores desvían parte de la tasa de muestreo predicha por el paralelismo implícito a otros esquemas, no necesariamente representados en la población de padres.

Se produce la **disrupción** de un esquema cuando a la salida de uno de los operadores, éste desaparece a pesar de haber estado representado en la entrada. El caso contrario a la disrupción es la **generación**.

Incluyendo el efecto del cruce en la Ec. (3.1), para una probabilidad de cruce p_c entre dos genotipos, uno escogido de forma proporcional a su adaptación, y el otro con probabilidad uniforme[3],

$$\begin{aligned} M(H,\ t+1) = &(1-p_c)M(H,\ t)\frac{f(H,\ t)}{\bar{f}} \\ &+ p_c\left[M(H,\ t)\frac{f(H,\ t)}{\bar{f}}(1-l_c)+G_c\right] \end{aligned} \tag{3.4}$$

donde l_c y G_c son, respectivamente, las pérdidas de disrupción y la ganancia de cruce de los esquemas. Para describirlas, vamos a considerar los tres casos posibles de cruce, donde se puede producir disrupción o no, o aparecer un nuevo esquema. Nos fijamos en el hiperplano o esquema $H_{12} = [10*]$ del ejemplo anterior.

Cruce sin disrupción: Supongamos que la selección nos entrega los individuos $g_0 = [100]$ y $g_1 = [001]$. La probabilidad de cruce es $p_c = 0{,}70$, y en una tirada aleatoria obtenemos 0,34, por lo que se aplica la recombinación. El punto de cruce se encuentra entre el primer y segundo elementos; g_0 es una muestra de H_{12}, y g_1 no lo es. El cruce genera dos hijos, $g_2 = [101]$ y $g_3 = [000]$; no ha habido disrupción del esquema $[10*]$ porque al menos uno de los hijos pertenece también al hiperplano H_{12}. Estos casos hacen bajar el valor de l_c.

Cruce con disrupción: Si el cruce se hubiera producido, en cambio, entre los genotipos $g_0 = [100]$ y $g_4 = [011]$, los hijos serían $g_5 = [111]$ y $g_3 = [000]$ con lo que H_{12} habría perdido un representante en la población. Estos casos hacen subir el valor de l_c.

Cruce con generación: En el caso de que la selección proponga como candidatos a $g_5 = [111]$ y $g_3 = [000]$, y que la recombinación se lleve a

[3]Por razones históricas [25] se suele seguir este desarrollo, y luego modificar más adelante la expresión para el caso en que los dos padres se seleccionen de forma proporcional a la adaptación

cabo como en los puntos anteriores, obtenemos como hijos los genotipos $g_0 = [100]$ y $g_4 = [011]$. Mientras que ni g_5 ni g_3 son muestras de H_{12}, uno de los hijos sí lo es. Estos casos determinan la cantidad G_c.

De esta forma, de entre todos los cruces producidos dentro de la distancia de definición, una fracción l_c de los mismos producirá disrupción, mientras que una cantidad G_c de genotipos aparecerá en el hiperplano H_{12}.

En este punto de la teoría de los esquemas se hace una suposición conservadora para acotar superiormente este efecto.

Definición 3.4 (Peor caso de disrupción de esquemas por cruce) *Se considera que cualquier punto de cruce dentro de la distancia de definición del esquema causa una disrupción.*

Aplicando la Def. 3.4 a la Ec. (3.5) obtenemos

$$\begin{aligned} M(H,\ t+1) \geq & (1-p_c) M(H,\ t) \frac{f(H,\ t)}{\bar{f}} \\ & + p_c \left[M(H,\ t) \frac{f(H,\ t)}{\bar{f}} (1 - l_d) \right] \end{aligned} \tag{3.5}$$

donde l_d, las **pérdidas por disrupción**, son el caso peor de las pérdidas l_c (ya que no incluyen los cruces sin disrupción).

Consideramos ahora una excepción a la Def. 3.4, que consiste en incluir el efecto del cruce de dos individuos del esquema H en cuestión, un subconjunto de los casos de cruce sin disrupción.

Definición 3.5 (Representación propocional de un hiperplano) *Se llama* representación proporcional de un hiperplano H *a la cantidad*

$$P(H,\ t) = \frac{M(H,\ t)}{M} \tag{3.6}$$

donde M es el tamaño de la población. Esta cantidad es la probabilidad de que un genotipo escogido al azar sea una muestra de H.

Para el cruce por un punto, esta cota algo menor de la disrupción viene dada por la igualdad

$$l_d = \frac{\Delta(H)}{L-1} (1 - P(H,\ t)) \tag{3.7}$$

Dividiendo entre el tamaño de la población, y con la Ec. (3.7), podemos simplificar (3.5).

$$P(H,\ t+1) \geq P(H,\ t)\frac{f(H,\ t)}{\bar{f}}\left[1 - p_c\frac{\Delta(H)}{L-1}\left(1 - P(H,\ t)\right)\right] \tag{3.8}$$

Si consideramos ahora que también el segundo padre se selecciona de forma proporcional a su adaptación,

$$\begin{aligned} P(H,\ t+1) \geq & P(H,\ t)\frac{f(H,\ t)}{\bar{f}} \cdot \\ & \left[1 - p_c\frac{\Delta(H)}{L-1}\left(1 - P(H,\ t)\frac{f(H,\ t)}{\bar{f}}\right)\right] \end{aligned} \tag{3.9}$$

Falta por último añadir el efecto de la mutación, que definimos gen a gen con una probabilidad de negación del valor p_{neg} sobre un esquema H de orden $o(H)$ (para que la mutación produzca disrupción, tiene que cambiar el valor de todos los genes fijos del esquema). Así llegamos a la expresión

$$\begin{aligned} P(H,\ t+1) \geq & P(H,\ t)\frac{f(H,\ t)}{\bar{f}} \cdot \\ & \left[1 - p_c\frac{\Delta(H)}{L-1}\left(1 - P(H,\ t)\frac{f(H,\ t)}{\bar{f}}\right)\right] \cdot \\ & (1 - p_{\text{neg}})^{o(H)} \end{aligned} \tag{3.10}$$

Teorema 3.6 (Teorema de los esquemas) *Los esquemas mejores que la media, de orden bajo, y cortos, aumentan su presencia en la población de forma exponencial en generaciones sucesivas.*

3.5.2. HBC (Hipótesis de los bloques de construcción)

Como hemos visto en el Teorema 3.6, los esquemas cortos, de orden bajo y valor de adaptación alto, prosperan en la población. Estos esquemas se recombinarán con otros para formar esquemas más largos y de orden más alto que potencialmente pueden tener valores de adaptación mayores, por lo que se llaman **bloques de construcción**.

La hipótesis la formuló Goldberg [18]: *Los esquemas muy adaptados, de orden bajo, y cortos, se muestrean, recombinan y se vuelven a muestrear para formar cadenas que potencialmente tienen un valor de adaptación mayor.*

La idea que hay detrás de todo esto es intuitiva y funciona con problemas de la vida real: Los problemas complejos se resuelven descomponiéndolos en otros más simples, resolviendo éstos y combinando las soluciones.

3.6. Dificultad en los AG

Dada una codificación canónica de longitud L del genotipo de los individuos de un algoritmo genético, si la epistasis o dependencia no lineal de los genes es cero, el problema se reduce a la optimización de L espacios de dimensión 1 y tamaño n, si los alelos tienen n valores posibles, ya que cada gen se puede optimizar por separado (lo que se conoce como método escalador).

Éste no es el caso general, por lo que el espacio de evolución[4] es un hipercubo de tamaño n^L ($n = 2$ en el caso binario, y la búsqueda se produce en las aristas del hipercubo).

Para este tipo de problemas, la búsqueda aleatoria o por fuerza bruta (la búsqueda exhaustiva donde se evalúan todos los puntos del espacio discreto) no tiene sentido. Pero si no podemos ser exhaustivo, y salimos también de la búsqueda aleatoria, estamos introduciendo un sesgo en la búsqueda, lo que se traduce por una preferencia por determinadas zonas del espacio de evolución.

Los algoritmos genéticos, frente a otros métodos de optimización, son un compromiso entre una búsqueda *fuerte* y una búsqueda *débil*. La búsqueda es mucho más fuerte que en el caso aleatorio, pero a la vez demasiado débil como para poder demostrar que se ha alcanzado un óptimo global. Este tipo de métodos se conocen como **métodos débiles**, y son métodos robustos y generales, porque trabajan con muy poco información previa sobre el problema.

A los algoritmos genéticos se los considera métodos de búsqueda global que no emplean la información del gradiente [54], por lo que son apropiados para optimizar funciones no diferenciables o funciones con profusión de mínimos locales.

Esta generalidad tiene su contrapartida en que pueden funcionar peor que los métodos especializados en problemas concretos. Por eso se emplean en ocasiones (en este Proyecto, por ejemplo), como se ve en la sección 3.4 sobre algoritmos híbridos y en el diagrama de la Fig. 3.1, en combinación con un algoritmo de búsqueda local específico para el problema.

Tradicionalmente se ha relativizado la validez y utilidad del teorema de los esquemas desde dos puntos de vista

- Es una inecuación, por lo que sólo representa un límite. Si se calcula el valor de adaptación medio de un esquema en un instante t, $f(H)$, la predicción del teorema de los esquemas sobre el valor de $f(H)$ en generaciones sucesivas puede ser imprecisa o incluso engañosa [21, 54].

[4] Aunque en [54] y otros se habla de *espacio de búsqueda*, es más apropiado el término *espacio de evolución*, según el diagrama de la Fig. 3.1, ya que es donde se "mueve" el algoritmo genético, y en la mayoría de los casos no hay una biyección entre ambos.

- El teorema de los esquemas habla de cada hiperplano *fuera* de contexto. Sin embargo, a medida que el algoritmo avanza, las muestras de un esquema en la población dejan de estar distribuidas uniformemente sobre éste, a causa de la orientación que sufren hacia (esperemos) los mejores hiperplanos; este efecto hace que la predicción sólo sea válida, a lo sumo, para una o dos generaciones [22]. No se trata sólo de incluir el factor tiempo, usando $f(H,\ t)$ [54, 54] en lugar de $f(H)$ [18], sino darse cuenta de que la distribución de muestras sobre H deja de ser uniforme, por lo que el valor de adaptación medio no se puede estimar correctamente.

Thornton [52] llega incluso a la conclusión de que *el AG sólo funciona bien en las situaciones en las que está garantizado que no funciona con efectividad*, proponiendo que los presupuestos del teorema de los esquemas contradicen a los de la HBC. Desde nuestro punto de vista, este autor no sólo no tiene en cuenta la complejidad del efecto del cruce para una representación y problema dados (supone que el cruce es negativo siempre), sino que además equivoca la función de los BC en la búsqueda de soluciones (supone que la información de los bloques sólo se añade por concatenación, y no por intersección de hiperespacios, por lo que sólo funcionan los bloques largos).

Goldberg [18] adelantaba en 1989 que había pruebas empíricas, sobre distintos problemas, que sostienen la HBC, citando [4, 41, 23, 24]. Desde entonces hasta ahora [20] (de hecho, no hemos conseguido esta referencia, a la que hemos llegado a través de Rothlauf [43]) se ha desarrollado bastante la teoría de la dificultad de los AG.

3.6.1. Clases de dificultad

Siguiendo la exposición de [43], hay dos factores esenciales para determinar la dificultad de un problema de optimización: la representación [36] y el método de optimización.

Goldberg [20] define tres tipos de dificultad para los BC, basándose en el teorema de los esquemas, en problemas de cruce y recombinación.

Intra-BC: Según el engaño asociado a la función de evaluación, que consiste en que entre varios esquemas que compiten por obtener muestras, uno o varios de ellos, que no contienen al óptimo, tienen mejor valor de adaptación medio que otro que sí lo contiene.

Inter-BC: Por una contribución no homogénea de todos los BC a la adaptación del problema, o por interdependencia entre los BC.

Extra-BC: Existe un ruido en el algoritmo generado por la selección aleatoria basada en adaptación, e incluso el valor de evaluación de los individuos puede no ser estacionario. Esto da lugar a la **deriva genética**.

Un análisis minucioso de esta teoría sobrepasa ampliamente el ámbito de este Proyecto, por lo que remitimos al lector interesado a las referencias mencionadas [20, 43]. Sí que es necesario, en cambio, dar una idea de los problemas principales para justificar las decisiones de diseño del método que sigue este Proyecto.

En concreto, necesitamos distinguir el efecto de la epistasis y del engaño respecto de la HBC. Cuando hay una epistasis muy alta, la HBC no es válida, ya que la recombinación de bloques da saltos aleatorios (desde nuestro punto de vista, ya que desconocemos la estructura de la epistasis) sobre la función de evaluación. En cambio, en las funciones engañosas, la HBC se puede estar cumpliendo en el sentido de que el algoritmo puede obtener mejores soluciones al combinar los bloques, pero de forma que se aproxima a un atractor local no óptimo.

3.6.2. Disrupción por cruce (epistasis)

Cuando en la representación existe epistasis (no linearidad entre genes), esto quiere decir que hay bloques de varios bits relacionados que contienen información. Si estos bloques son largos (independientemente del orden que tengan, aunque la longitud del bloque no puede ser menor que su orden), la disrupción provocada por el cruce es muy negativa, la HBC no se cumple, y la búsqueda se convierte en casi aleatoria.

Desde el punto de vista del teorema de los esquemas, al no cumplirse la condición de bloques cortos, los esquemas se parten y no reciben un gran número de muestras.

Si toda la información está en bloques de orden uno (si la epistasis es cero o, equivalentemente, el problema es separable linealmente), el problema es sencillo para el AG. En realidad, el cruce no tiene efecto, y el algoritmo se convierte en un escalador que puede encontrar el óptimo por medio de mutación gen a gen.

Hacemos notar que la no linearidad de la representación no tiene que ver con la no linearidad de la función de evaluación. Por ejemplo [43],

$$f(x) = \begin{cases} 1 & \text{si } x = 00, \\ 2 & \text{si } x = 01, \\ 4 & \text{si } x = 10, \\ 10 & \text{si } x = 11, \end{cases} \tag{3.11}$$

Esta función no es lineal y, sin embargo, sus bloques tienen distancia 0 (orden 1).

En este Proyecto se lucha contra este efecto de disrupción. Mediante un análisis previo, se localizan los bloques, y mediante una transformación lineal se acortan sin pérdida de información. Por supuesto, si consiguiéramos llevar los bloques a distancia 0 (orden 1), no sería necesario el cruce, pero como no es así, lo mantenemos. Se explicará este proceso con más detalle más adelante.

3.6.3. Deriva genética

La deriva genética es una consecuencia del comportamiento no determinista de la selección respecto del valor de adaptación, del efecto de la mutación, y de la no estacionariedad del valor de adaptación de los individuos.

Se ha demostrado [44] que a menos que se use elitismo, el AG no puede converger a la solución óptima, aunque sí puede encontrarla. El elitismo puede ocasionar por otra parte una convergencia prematura, lo que no es deseable, pero en los problemas de optimización es una opción habitual, por lo que lo incluimos en nuestro algoritmo.

3.6.4. Dificultad desde el punto de vista del espectro

Un análisis muy utilizado para medir la dificultad del problema para el AG es por los coeficientes de Walsh. Para una función $f : \{0,\ 1\}^L \mapsto \mathbb{R}$ siempre existe una descomposición

$$f(x) = \sum_{j=0}^{2^L-1} w_j \psi_j(x) \tag{3.12}$$

Las funciones de Walsh son funciones $\psi : \{0,\ 1\}^L \mapsto \{1,\ -1\}$

$$\psi_j(x) = (-1)^{cb(j \wedge x)} \tag{3.13}$$

donde cb(x) es el número de "1"s en x, y $\wedge$ es el operador "Y lógico" bit a bit. Los **coeficientes de Walsh** w_j son de la forma

$$w_j = \frac{1}{2^L} \sum_{i=0}^{2^L-1} f(i)\psi_j(i) \tag{3.14}$$

El orden del coeficiente de Walsh w_j se define como $cb(w_j)$. Por ejemplo, para $L = 3$, un coeficiente de orden uno, w_{100}, describe el efecto lineal del tercer bit en la función de evaluación; un coeficiente de orden dos, w_{101}, describe la interacción cuadrática entre el primer y tercer bits; y el de orden 3, w_{111}, la interacción cúbica entre todos los bits de la representación.

Se ha demostrado [36] que para cualquier función con un único óptimo sobre un espacio discreto existe una representación para la que el problema es sencillo, desde el punto de vista de los engaños, y que para cualquier función engañosa, existe una transformación[5] T tal que $T(f)$ es sencilla.

Relacionado con esta afirmación, encontramos que un problema no es inicialmente fácil o difícil para el AG, y que esta dificultad depende de la representación y del método [43].

Basándose en una definición de espectro a partir de los coeficientes de Walsh, Salcedo [45] ha estudiado el efecto de las transformaciones lineales unitarias sobre la dificultad del algoritmo. Se puede demostrar que una transformación lineal unitaria[6] (TLU) aleatoria, en general empeora sus prestaciones, pero que una transformación lineal unitaria bien escogida puede mover L particiones linealmente independientes a cualquier orden (nos interesa que éste orden sea 1). Necesitamos ahora definir qué es una partición.

Definición 3.7 (Partición) *Se llama* partición *a un vector* $H_i \in \{d, *\}^L$ *que representa a todos los esquemas que coinciden con ella en sus posiciones "$*$", y que tienen bits fijos en las posiciones "d". En ocasiones se representa la partición transformando las "d" en "1" y los "$*$" en "0", pero no deben confundirse con los esquemas.*

Definición 3.8 (Orden de la partición) *Se llama* orden de la partición *al número de posiciones "d" de ésta.*

Por ejemplo, la partición [d $* * *$ d] contiene a los esquemas [0 $* * *$ 0], [0 $* * *$ 1], [1 $* * *$ 0] y [1 $* * *$ 1]. Tiene orden 2, y se puede representar como [1 0 0 0 1], siempre que no perdamos de vista que es notación de partición y no de esquema.

El interés de las TLU radica en que su aplicación no modifica los valores de los coeficientes de Walsh (sólo las particiones asociadas a estos), por lo que no hay pérdida de información [45].

Sin embargo hay algunos puntos a tener en cuenta con este análisis, que pueden limitar la mejora en las prestaciones que se persigue con este trabajo. Aunque en [45] la teoría se desarrolla para particiones, y nosotros trabajamos desde el punto de vista de genotipos, implícitamente estamos moviendo particiones, por lo que podemos aprovechar la teoría.

[5] Nótese que "transformación" no implica "transformación lineal"

[6] Una transformación lineal $T(x) = Ax$ es unitaria sii $\det(A) = 1$, lo que es equivalente a decir que la matriz es invertible.

- En una representación $\{0,\ 1\}^L$ sólo puede haber L particiones linealmente independientes. Una transformación lineal unitaria puede llevar L particiones linealmente independientes a otras L particiones linealmente independientes [45] a orden 1 (longitud de los bloques igual a cero). Si toda la información está contenida en estas particiones, el problema se resuelve fácilmente con el AG.

- En [45] se advierte que si la partición es la combinación lineal de n particiones (si $n > 1$ la partición es linealmente dependiente), el orden al que se puede llevar con una transformación lineal unitaria es n. Dada una representación en $\{0,\ 1\}^L$, sólo puede haber L particiones linealmente independientes, que forman una base. El resto son linealmente dependientes, y en media se irán a órdenes medios ($L/2$), lo que es negativo. De hecho, si la información se encuentra repartida uniformemente en muchas particiones, una TLU seguramente estropee las prestaciones del AG.

- El orden más alto entre los coeficientes de Walsh de una función sólo da una idea de la cota superior de la complejidad del problema [43]. En el ejemplo de la Ec. (3.11), el problema se puede optimizar fácilmente por escalada, pero los coeficientes de Walsh son $w = \{4{,}25,\ -1{,}75,\ -2{,}75,\ 1{,}25\}$.

- Los experimentos de Salcedo [45], a partir de una definición del espectro que agrupa los coeficientes de Walsh según el orden de las particiones a las que están asociados, sugieren que si todos los coeficientes sólo están en órdenes altos, el problema es muy difícil para el AG.

- También parece que si el espectro sufre una compresión, el rendimiento del AG mejora (al margen de que fuera ya bueno o malo), por lo que incluso aunque una medida del espectro no aporte información definitiva *per se*, sí podría ser significativa la distribución relativa del mismo antes y después de una transformación lineal.

3.6.5. Engaños

Como vimos en la clasificación de dificultades, si la función de evaluación es engañosa, el algoritmo se esfuerza por explorar hiperplanos donde no hay óptimos globales.

Una solución relativa [43] consiste en aumentar el tamaño de la población esperando así que los hiperplanos mejores, que a causa del engaño no lo parecen, reciban suficientes muestras. Sin embargo, la solución adecuada está en un cambio de representación.

Hay varios puntos a tener en cuenta a la hora de abordar las funciones engañosas:

- Si bien es cierto que para funciones donde los bloques tienen distancia 0 (orden 1), no existe engaño ni epistasis, Goldberg [19] ha demostrado que se pueden construir funciones parcialmente engañosas con coeficientes de Walsh de sólo orden 2, y totalmente engañosas con coeficientes de orden 3.

Actualmente se trabaja con transformaciones lineales, tanto inversiones como transformaciones generales. Se ha demostrado sobre funciones unimodales [36] que la inversión no es lo suficientemente potente como para evitar los engaños, mientras que al menos las transformaciones afines sobre el grupo diádico[7] pueden cambiar algunos problemas engañosos por otros sencillos. Es este el camino que seguiremos en este Proyecto.

Parecería lógico hacer un análisis espectral [45], a la hora de predecir los resultados experimentales. Alternativamente se puede hacer una medida de la epistasis [9]. Sin embargo, esto supera el ámbito del Proyecto; para el cálculo del espectro y de la varianza, es necesario conocer toda la función de evaluación (esto por supuesto es absurdo desde el punto de vista del que quiere sólo resolver el problema, ya que supone una búsqueda exhaustiva, aunque tiene justificación para el teórico que quiere comprobar cómo se resuelve el problema). Para un problema mínimamente representativo, la longitud del genotipo está en torno a 75 bits. El espacio de evolución tiene entonces un tamaño de $2^75 \simeq 3{,}8 \cdot 10^{22}$. Si el cálculo de cada individuo llevara sólo un segundo (en las máquinas de que disponemos este tiempo suele ser superior en al menos un orden de magnitud), el tiempo de cómputo necesario total sería de unos $3{,}6 \cdot 10^{16}$ años. Los últimos datos que tenemos sobre la edad del Universo apuntan a unos $1{,}4 \cdot 10^{10}$ años. Incluso con el problema más pequeño del que disponemos, $L = 25$, necesitaríamos cerca de 1 año de cálculo.

Descartado el cálculo, podemos pensar en la estimación. Conocemos métodos de estimación del espectro y de la epistasis [9, 45], pero sus autores reconocen falta de precisión en los resultados. Por lo tanto, no consideramos razonable respaldar unos resultados con unos métodos que todavía no están maduros.

Incluso con buenos estimadores del espectro, las anotaciones de este apartado, junto con las del apartado anterior, nos hacen dudar de la efectividad del análisis de Walsh para predecir las prestaciones del AG.

[7] Una transformación afín sobre el grupo diádico G es una proyección $f : G^N \mapsto G^N$ de la forma $f(p) = A \cdot p + q$, donde $p, q \in G^N$ y A es una transformación lineal en G^N.

Capítulo 4

Diseño del algoritmo

4.1. Decisiones de diseño

4.1.1. Representación

En este Proyecto se utiliza una representación muy similar a la de Esbensen [14], hibridando un algoritmo genético simple con el HRD (Heurístico de Red de Distancias). Para una instancia del problema de Steiner definida sobre un grafo con n vértices, de los cuales queremos abarcar m con un árbol, el genotipo consiste en una cadena de $n - m$ bits. Cada instancia del problema está descrita en un fichero de texto; por ejemplo, para el grafo ilustrativo del artículo de Esbensen que se puede ver en la Fig. 4.1, es el del Cuadro 4.1 (a la derecha, las $\rightarrow$ señalan el significado de cada línea del fichero).

La codificación puede resumirse en tres puntos:

- Cada bit del genotipo está asociado a uno de los vértices $v \in R = V \backslash W$.
- Un bit con valor "1" indica que el vértice representado por él necesariamente será un vértice de Steiner en la descodificación.
- Un bit con valor "0" indica que el vértice representado por él podrá ser o no un vértice de Steiner en la descodificación, dependiendo de las decisiones que tome el HRD.

La razón de que los vértices de W no se codifiquen es que siempre tendrían el valor 1. Para el ejemplo de la Fig. 4.1 la codificación es la de la Fig. 4.2.

Contrastamos a continuación la representación escogida, según la recopilación de Rothlauf [43]. En primer lugar, Goldberg [18] da dos recomendaciones para la representación:

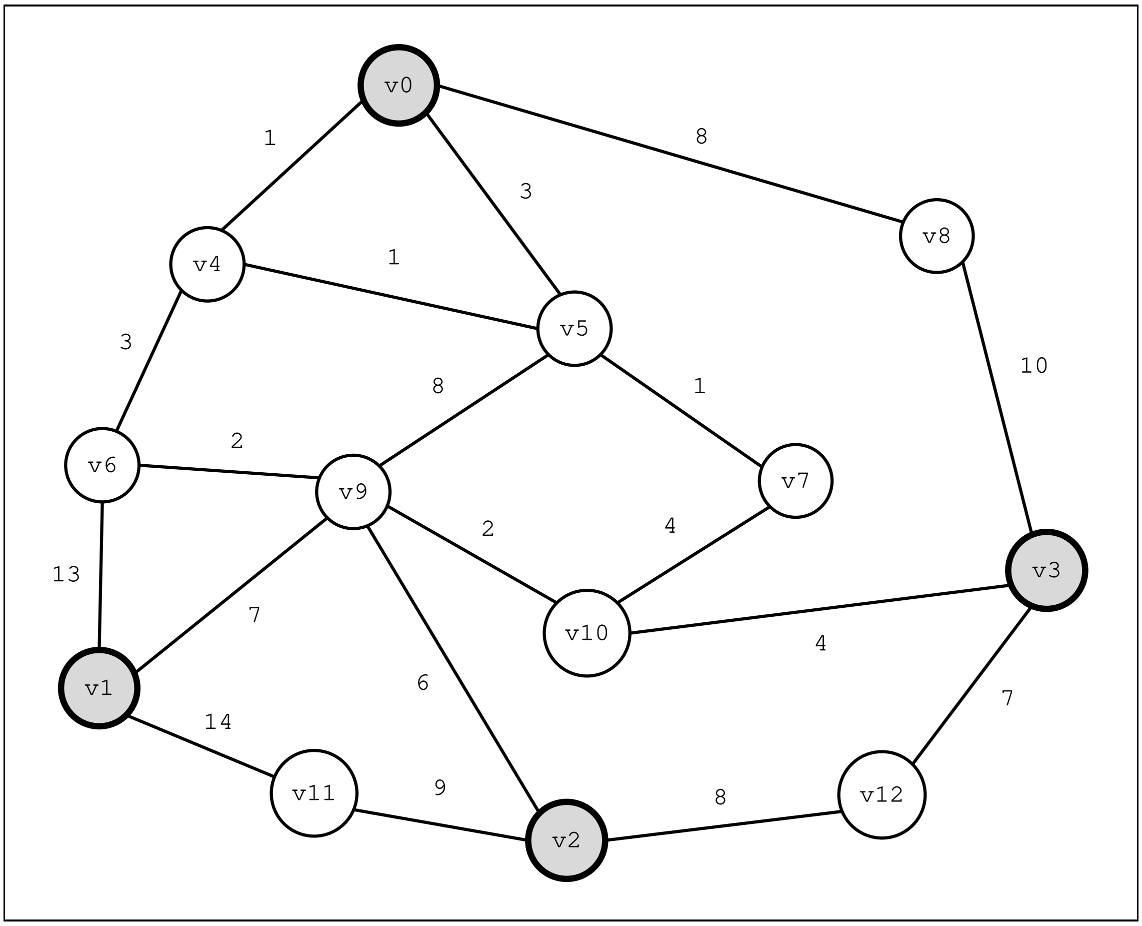


Figura 4.1: Grafo de Esbensen.

13	19			$\rightarrow$	nº de vértices y aristas
0	8	8		$\rightarrow$	e_0
8	3	10		$\rightarrow$	e_1
0	5	3		$\rightarrow$	e_2
5	7	1		$\rightarrow$	e_3
0	4	1		$\rightarrow$	e_4
4	5	1		$\rightarrow$	e_5
9	5	8		$\rightarrow$	e_6
6	4	3		$\rightarrow$	e_7
6	9	2		$\rightarrow$	e_8
7	10	4		$\rightarrow$	e_9
9	10	2		$\rightarrow$	e_{10}
6	1	13		$\rightarrow$	e_{11}
1	11	14		$\rightarrow$	e_{12}
11	2	9		$\rightarrow$	e_{13}
2	9	6		$\rightarrow$	e_{14}
10	3	4		$\rightarrow$	e_{15}
12	3	7		$\rightarrow$	e_{16}
2	12	8		$\rightarrow$	e_{17}
1	9	7		$\rightarrow$	e_{18}
4				$\rightarrow$	nº de vértices obligatorios
0	1	2	3	$\rightarrow$	$w_0,\ w_1,\ w_2,\ w_3$

Cuadro 4.1: Ejemplo comentado de un fichero con los datos de un problema.

v4	v5	v6	v7	v8	v9	v10	v11	v12

Figura 4.2: Codificación para el ejemplo de Esbensen.

1. Principio de los bloques de construcción significativos: Los esquemas deben ser cortos, de orden bajo, y con poca correlación con esquemas de otras particiones.

2. Principio de los alfabetos mínimos: El alfabeto de la codificación debe ser lo más pequeño posible mientras no interfiera con la representación natural de las soluciones.

Asumimos la validez de la HBC y del teorema de los esquema con la primera recomendación, y por lo tanto también la segunda, ya que para una cardinalidad mínima, el número de esquemas es máximo, por lo que es más fácil buscar similitudes entre las cadenas.

Se puede objetar que la representación natural de las soluciones de un grafo no es una cadena de bits, sino las estructuras de datos que representan árboles directamente. Ya hablamos de las codificaciones *ad hoc* en la sección 3.3.3, y cómo su inconvente es tener que desarrollar operadores específicos, además de dificultar el análisis teórico. Si además recordamos [36] que para cualquier función f (monomodal) definida en un espacio discreto, existe una representación tal que f es sencilla, y que ya existe una gran cantidad de teoría desarrollada para algoritmos genéticos, podemos considerar que la importancia de avanzar más en la búsqueda de transformaciones sobre representaciones binarias que generen problemas sencillos, supera a la de encontrar un método particular eficiente para resolver un problema dado, ya que cualquier resultado obtenido en este ámbito es válido para cualquier tipo de problema.

El segundo grupo de recomendaciones de Rothlauf hace referencia a los estudios de Palmer [38] sobre la representación de árboles, que se puede extender a otras representaciones.

1. La codificación debe representar todos los fenotipos posibles. En este sentido, la representación de Esbensen es insuficiente, porque el HRD trabaja restringiendo el espacio de búsqueda a un subespacio prometedor, en función del genotipo, de forma determinista, y resuelve los empates de forma arbitraria – que no aleatoria (para esto último, el

genotipo debe guardar además la semilla aleatoria, para que aunque pueda haber más de un fenotipo asociado para unos valores de los bits del genotipo dados, que al menos sea siempre el mismo para el mismo individuo). Sin embargo, esta circunstancia radica en la idea misma de algoritmo genético híbrido, y sólo es mejorable el aspecto de resolución aleatoria de los empates, que por razones de simplicidad en la implementación no se ha hecho[1]. Un poco más adelante podrá verse en el ejemplo de funcionamiento de HRD qué problemas ocasiona esta decisión.

2. La codificación debe ser insesgada en el sentido de que todos los individuos estén representados igualmente en el conjunto de todos los genotipos. Esto evidentemente no se cumple con la codificación de Esbensen, ya que cada árbol solución tiene asociados al menos todos los genotipos que tienen entre el número mínimo de "1"s necesarios para generarla, y todos los que añaden a éste "1"s en las posiciones de los vértices de Steiner del árbol solución; no todos los árboles tienen el mismo número de vértices, por lo que no todos los fentipos tienen la misma redundancia.

3. Una codificación no debe representar soluciones erróneas. Como vimos anteriormente, es posible permitir las soluciones erróneas y penalizarlas o repararlas, pero esto tiene dos inconvenientes:

   - Si se permiten y penalizan, esto introduce fuertes discontinuidades en la función de adaptación y hace que el AG funcione peor, como en [28].
   - Si se reparan, esto supone una carga computacional añadida, además de tener que ser muy cuidadoso con la función de reparación (que consiste en conectar componentes y eliminar ciclos) para que no introduzca un sesgo añadido.

   La magnitud del problema de las soluciones no válidas es muy grave cuando trabajamos con árboles, ya que la probabilidad de generar árboles válidos aleatoriamente disminuye exponencialmente con el número de vértices [38] – aunque si confiamos en la HBC, la generación de árboles entre generaciones no sería aleatoria, pero este desarrollo no está estudiado. Este aspecto es el que hace más interesante la representación que usamos, ya que dado un conjunto de vértices, el HRD *siempre* proporciona una solución válida.

[1]Esbensen no aclara si su implementación resuelve los empates de forma arbitraria o aleatoria.

4. La descodificación del fenotipo a partir de genotipo debe ser sencilla. Esta es otra razón por la que el algoritmo local que usamos es el HRD, que para un grafo de n vértices, m de los cuales queremos abarcar, la complejidad temporal es $\mathcal{O}(mn^2)$.

5. La codificación debe tener localidad, esto es, que cambios pequeños en el genotipo resulten en cambios pequeños en el fenotipo. Esto, si nos damos cuenta, de alguna forma es equivalente a decir que la representación permita una separación del problema en bloques relativamente independientes entre ellos, o que se den las condiciones para la HBC, lo que es dependiente de la topología de cada grafo.

Por último es necesario añadir una consideración propia sobre la representación. En nuestra codificación, un "0" no tiene la misma información que un "1", siendo estos últimos mucho más importantes. Un "1" significa que el vértice correspondiente está *obligatoriamente* en la solución, mientras que un "0" indica que puede estar o que no. Además, según el resultado por el que el número máximo de vértices de Steiner es $s \leq \text{mín}(m-2,\ r)$, los vectores serán típicamente huecos, con muchos más ceros que unos.

Como veremos más adelante, esto nos plantea un problema a la hora de seleccionar los esquemas que queremos "proteger", por lo que habrá que estudiar si deben tener preferencia las posiciones más deterministas, sean éstas de "0"s o "1"s indistintamente, o las posiciones de "1"s, aunque sean algunas menos deterministas que otras de "0"s.

4.2. Descripción del HRD.

Vamos a estudiar ahora el heurístico que se encarga de la descodificación del genotipo. Nos interesa seguir el mismo ejemplo que Esbensen [14], que se corresponde con el del genotipo cuyos bits son todos "0" (sólo es obligatorio que estén en la solución los vértices de W), ya que así podremos enseñar además el efecto de la arbitrariedad en los empates. Para generalizar a otros genotipos, sólo hay que marcar los vértices cuyos bits correspondientes sean "1", como si estuvieran originalmente en el problema definidos como vértices $w \in W$, y seguir exactamente los mismos pasos.

En los apartados siguientes se presenta el diseño desde un punto de vista funcional; para una descripción de los algoritmos empleados en la implementación se puede consultar el Cap. 5.

4.2.1. Cálculo del grafo de distancias $D(G)$

En general el grafo G del problema no estará completamente conectado (esto es cierto en todas las instancias de la biblioteca OR [5]), por lo que se construye un nuevo grafo, $D(G)$, que sí lo está, de forma que la arista e_{ij} entre cada par de vértices $i,\ j$ tiene el peso del camino de menos distancia entre ellos en G.

Nótese que el hecho de que dos vértices estén unidos por una arista de peso $c_G(i,\ j)$ en G no quiere decir que en $D(G)$ sigan unidos por una arista del mismo peso en general, sino que tendrá peso $c_{D(G)}(i,\ j) \leq c_G(i,\ j)$.

También se debe tener en cuenta que junto con la distancia mínima, se guarda uno de los caminos que guarda esta distancia (en forma de matriz de predecesores), pero que es posible que exista más de uno, y que la decisión se toma de forma arbitraria.

4.2.2. Construcción del subgrafo G_1 de $D(G)$ inducido por W

G_1 se contruye eliminando los vértices de $D(G)$ que no están marcados en negrita, junto con sus aristas adyacentes, como se puede ver en la Fig. 4.3.

4.2.3. Cálculo de un árbol generador mínimo T_1 de G_1

En este apartado es donde se dan los empates más *peligrosos*, y la razón de que sigamos el mismo ejemplo que Esbensen. Analizando la Fig. 4.3, es evidente que existen cuatro árboles generadores mínimos con el mismo peso:

1. $e_{0,3}$, $e_{0,1}$, $e_{2,3}$.

2. $e_{0,3}$, $e_{0,1}$, $e_{0,2}$.

3. $e_{0,3}$, $e_{1,3}$, $e_{2,3}$.

4. $e_{0,3}$, $e_{1,3}$, $e_{0,2}$.

A partir de ahora vamos a representar simultáneamente los grafos que va creando la implementación de Esbensen (caso 1) y la nuestra (caso 2), que resuelven de forma distinta los empates. Los árboles T_1 de cada caso se pueden ver en la Fig. 4.4.

4.2.4. Expansión de T_1 en G_2

Se sustituyen las aristas de T_1 por las correspondientes al camino más corto que se guardó cuando se hizo el cálculo del grafo de distancias, Fig. 4.5.

4.2.5. Cálculo de un árbol generador mínimo T_2 de G_2

De nuevo se resuelven los empates, si los hubiera, de forma arbitraria, Fig. 4.6.

4.2.6. Poda de las ramas sobrantes

Cualquier arista que termine en un único vértice, si éste no está marcado en negrita, no es necesaria para el árbol, por lo que se eliminan de forma iterativa, hasta que no queda ninguna, Fig. 4.7.

Como se puede comprobar en la Fig. 4.7, el peso del primer árbol es 25 (el óptimo), y el del segundo, 29. Esto no quiere decir que la implementación de Esbensen sea mejor que la nuestra. Simplemente, en este caso particular ha dado la casualidad de que al deshacer arbitrariamente el empate del apartado 4.2.3, ha tenido más fortuna.

4.2.7. Diferencias de diseño con Esbensen [14]

Esbensen emplea algunos resultados de la teoría de grafos para la búsqueda de árboles generadores mínimos que le permiten reducir la complejidad del problema. Por medio de 4 tipos de reducciones, elimina del genotipo los vértices sobre cuya pertenencia o no a la solución no puede haber dudas, lo que lógicamente simplifica el problema. En este Proyecto, en cambio, el interés por el estudio de la mejora que puede aportar un cambio de representación a través de una transformación lineal es mayor que el de optimizar en términos absolutos el método para resolver los problemas de la biblioteca OR. Hemos considerado que la mejora sobre un grafo de una cierta dimensión n sin reducir estará relacionada con la de otro grafo con dimensión inicial $n' > n$, y al que las reducciones llevan a dimensión n, lo que puede ser aventurado, por el fuerte cambio de topología, pero basándonos en que el HRD da siempre soluciones válidas, y que el AGH con una buena representación debería ignorar estos vértices espurios. Por tanto, hemos supuesto que el no implementar las reducciones sólo supone una carga de computación añadida.

Hay otra buena razón para respaldar esta decisión. En un trabajo reciente [42, 33] se proponen instancias nuevas para el problema de Steiner en un grafo,

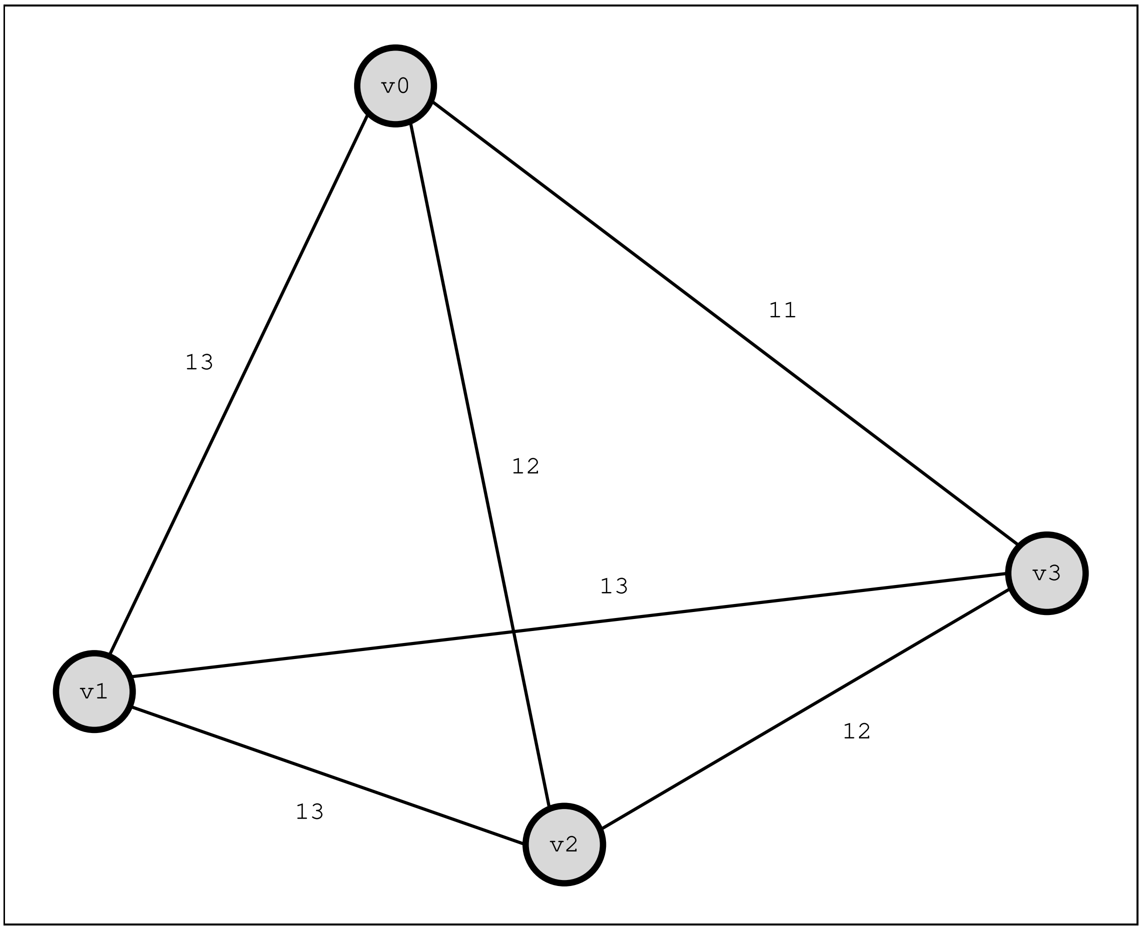
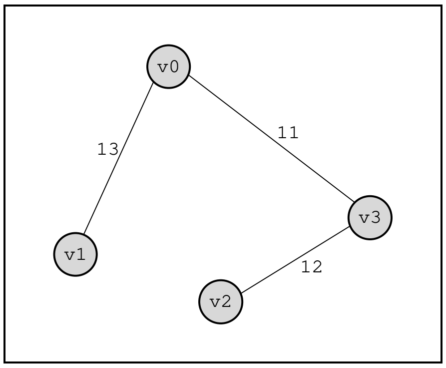
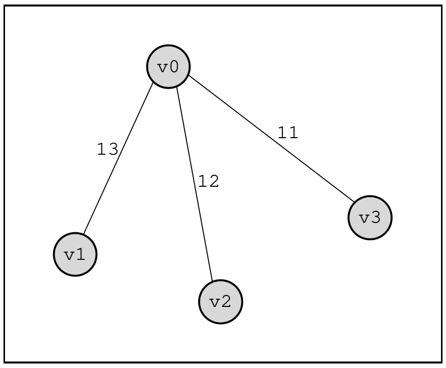


Figura 4.3: Grafo G_1 de $D(G)$ inducido por W.

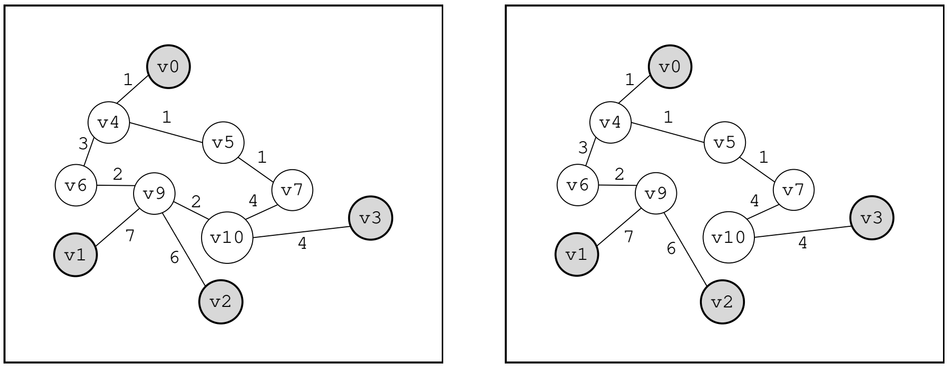


(a)



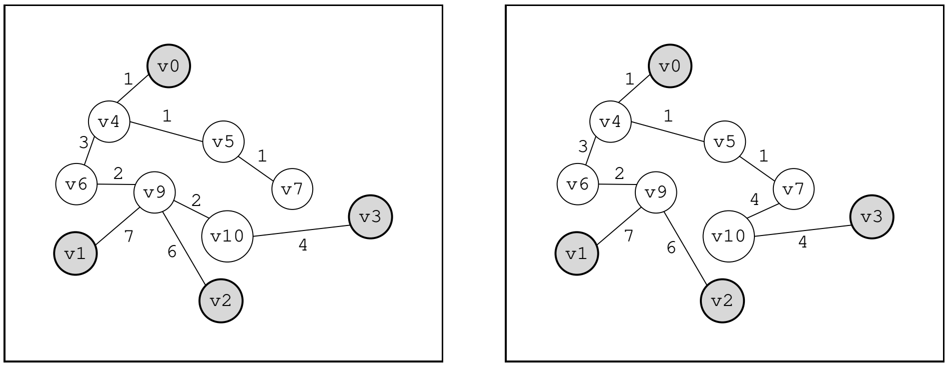
(b)

Figura 4.4: Árbol generador mínimo T_1



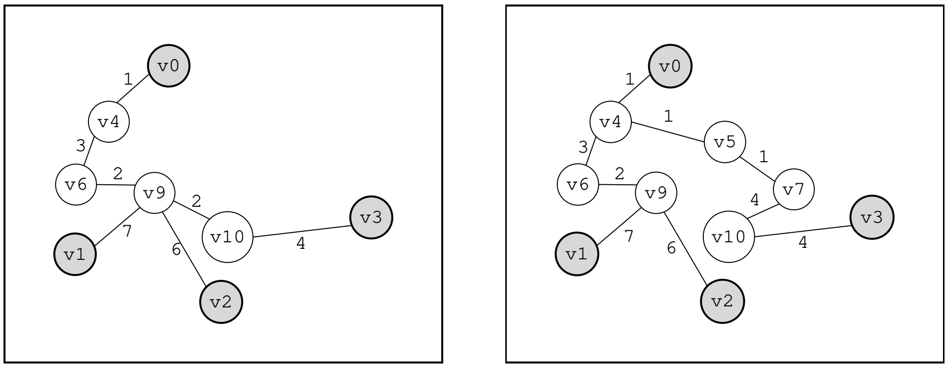
(a) (b)

Figura 4.5: Grafo expandido G_2.



(a) (b)

Figura 4.6: Árbol generador mínimo T_2.



(a) (b)

Figura 4.7: Árbol podado.

generadas con la idea de que sean resistentes a las reducciones.

La codificación de Esbensen se diferencia también de la de este Proyecto en el uso de la inversión. En aquella cada individuo tiene asociada una etiqueta que determina qué bit corresponde a cada vértice (como vimos en el apartado 3.3.10). Cuando dos individuos se tienen que cruzar, uno se reordena respecto al otro para que casen las posiciones. En cada generación, cada individuo tiene una probabilidad p_{inv} de invertirse, lo que se lleva a cabo seleccionandos aleatoriamente dos puntos y "dando la vuelta" a los bits comprendidos entre ellos (y a sus etiquetas).

Como vimos en el apartado 3.3.11, una inversión es un caso particular de transformación lineal, y es menos efectiva para mejorar la representación [36]. Esbensen asocia una inversión aleatoria a cada individuo. En este Proyecto, en cambio, definimos una transformación lineal única para todos los individuos y generaciones basándonos en el análisis de la evolución de las soluciones.

Por último, en el trabajo de Esbensen aparece un operador de mutación añadido que, basándose en el resultado de que el número de vértices de Steiner necesario es $|S| \leq \text{mín}(m-2,\ r)$, limita el número de "1"s en el genotipo. En este Proyecto se ha optado por renunciar a este conocimiento previo, y dejar que sea el propio AGH el que regule el número de "1"s.

4.3. Diseño de la transformación lineal.

En el campo de los AG, en concreto, hay dos corrientes básicas para abordar la búsqueda de la convergencia:

Bajar bloques a orden 1: Buscar que la información esté contenida en bloques de orden 1, reduciendo lo más posible la epistasis, y buscar las intersecciones de esquemas de orden bajo. En este caso se desaprovecha la capacidad de generar sinergia que predice la HBC, pero se puede llegar a las soluciones con cierta comodidad, y es un método robusto frente a los efectos negativos de la disrupción por cruce.

Aprovechar la epistasis: Tener conocimiento de la epistasis, y dirigir la HBC a partir de esta información. En este caso, la convergencia a soluciones óptimas sería más rápida, porque se aprovecharía el cruce sobre la HBC. El inconveniente es que determinar la epistasis puede ser un problema tan complejo como el que intentamos resolver.

En este Proyecto se sigue la primera opción, ya que no disponemos de las herramientas teóricas necesarias para abordar la segunda, y nos basamos en el trabajo de Salcedo [45], que hace lo mismo.

Resumiendo un poco las secciones anteriores, nos damos cuenta de que tenemos un problema con una codificación binaria donde los "1"s tienen más información que los "0"s, y un resultado [36] que nos indica que la transformación lineal es más potente que la inversión (que es sólo un caso particular) para mejorar la representación. También sabemos [45] que una transformación lineal unitaria (TLU) conserva toda la información. Estos son los parámetros que determinan el diseño de la transformación lineal (TL) que buscamos. Para este problema en particular, el diseño de la transformación no es inmediato. Así, vamos a tener que estudiar distintos tipos de estategias para el cálculo de ésta.

4.3.1. Análisis de esquemas para confeccionar una transformación

Podemos investigar la evolución del muestreo de hiperplanos del AGH calculando la probabilidad de aparición de "0"s y "1"s en cada posición del genotipo. Supongamos, por ejemplo, que en el grafo de la Fig. 4.1, tanto la posición v_4 como v_9 tienen una marcada tendencia a aparecer con un valor de "1", mientras que v_{12} casi siempre es "0".

Si aceptamos la HBC, podemos suponer que existe un esquema $[1 * * * *1 * *0]$ que contiene buenas soluciones. Este esquema tiene orden 3, pero su longitud de definición es de 8. Según el Teorema 3.6, será fácil que haya disrupción en este esquema, por ser tan largo.

Para localizar las posiciones importantes simplemente vamos acumulando el número de "1"s en cada posición, con dos variantes:

- Todos los individuos de la población, a lo largo de las generaciones.
- Todos los individuos de la población, sólo en la última generación.

En el segundo caso el significado de *última generación* depende de los criterios de parada. Desde un punto de vista práctico, no tiene sentido esperar a que el algoritmo converja para definir la TLU (si la solución ya es buena, no hace falta volver a buscarla), pero para poder estudiar si la definición de la TLU tiene sentido, es conveniente hacerlo así; luego, se puede estudiar si con un número pequeño de generaciones se puede estimar el mismo esquema.

Para decidir qué genes entran en el esquema, definimos un nivel de determinismo dado por

$$p(x_i) = 2\left(\frac{1}{PG}\sum_{j=0}^{P-1}\sum_{k=0}^{G-1} x_i(j,\ k) - 0{,}5\right) \tag{4.1}$$

donde P es el tamaño de la población, G el número de generaciones en las que se han tomado las estadísticas, y $x_i(j,\ k)$ la posición i del genotipo en el individuo j de la generación k.

La función de la Ec. (4.1) asigna un número real $[0{,}0, 1{,}0]$ a cada posición tal que el signo positivo (negativo) indica predominio de "1"s ("0"s), y el módulo, el grado de determinismo; 1,0 para bits que nunca cambian su valor, y 0,0 para bits aleatorios.

En caso de que la función sea multimodal, las muestras de la población se repartirán inicialmente entre los hiperplanos de varios óptimos. Sin embargo, en cuanto avance el algoritmo, las muestras se concentrarán en la región de uno de ellos. Esto no perjudica más al algoritmo con nuestra TL que sin ella; los métodos para evitarlo están orientados a aumentar la variedad genética, de forma que se mantengan varias soluciones al mismo tiempo. El lector interesado puede buscar información sobre métodos de nichos, como la adaptación compartida o el atestamiento.

Una vez localizado el esquema, tenemos que decidir cuál es la mejor relación entre posiciones fijas y "$*$", lo que estudiaremos experimentalmente, y cuál va a ser la TLU que va a proteger el esquema. Esto lo tratamos a continuación.

El método de protección que decidimos es frente a disrupción del cruce. Queremos pasar los esquemas largos a bloques de longitud 0. Por ejemplo:

$$\begin{bmatrix}1 & * & * & * & * & 1 & * & * & 0\end{bmatrix} \mapsto \begin{bmatrix}1 & * & * & * & * & * & * & * & *\end{bmatrix} \tag{4.2}$$

Con una inversión podríamos aspirar, a lo sumo, a llevar la longitud del bloque al mínimo permitido por su orden, pero no a cambiar su orden.

Lema 4.1 *Para un esquema de orden $o(H)$ sobre un genotipo de longitud L, el número máximo de vectores linealmente independientes que se puede definir es $L - o(H)$.*

Nos hemos encontrado con el primer toma y daca del diseño; podemos escoger varios esquemas de orden bajo para llevarlos a orden 1, o menos de orden más alto. Cuanto mayor sea el esquema, estaremos protegiendo una mayor cantidad de información, pero por otra parte también estaremos trabajando menos con la idea de que según la HBC, la solución se encuentra por el efecto combinado de bloques de orden bajo. Además, al aumentar el grado del esquema, las posiciones que añadamos serán cada vez menos deterministas, por lo tanto menos significativas; el efecto de proteger una posición que debe ser aleatoria en la búsqueda puede ser incluso contraproducente.

4.3.2. Matriz directa del análisis de perturbación mínima de la matriz identidad

De momento, plantearemos la matriz con la que vamos a implementar la TLU para el caso general. Para un esquema de orden $o(H)$ sobre un genotipo de longitud L, una forma de construir $L - o(H)$ vectores linealmente independiente, de forma que éstos sean lo más parecidos en orden al esquema es rellenar las posiciones "$*$" con una base canónica. Por ejemplo:

$$\begin{bmatrix} 1 & * & * & * & 1 & 0 & 1 \end{bmatrix} \mapsto \begin{cases} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \end{cases} \tag{4.3}$$

Ahora queremos pasar estos vectores a orden 1. Escogemos una base de destino que nos permita construir cómodamente la matriz identidad.

$$\begin{aligned} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} &\mapsto \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} &\mapsto \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} &\mapsto \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{4.4}$$

De esta manera, si llamamos T a la matriz identidad perturbada con los vectores del esquema que queremos *proteger*,

$$T = \begin{bmatrix}1\\1\\0\\0\\1\\0\\1\end{bmatrix} \begin{bmatrix}1\\0\\1\\0\\1\\0\\1\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\1\\0\\1\end{bmatrix} \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} = \begin{bmatrix}1&1&1&0&0&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&1&0&0&0\\1&1&1&0&1&0&0\\0&0&0&0&0&1&0\\1&1&1&0&0&0&1\end{bmatrix} \tag{4.5}$$

Buscamos entonces una transformación X que lleve los esquemas largos de T a los otros cortos.

$$X \cdot T = \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} = \begin{bmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix} \tag{4.6}$$

Evidentemente, de existir, $X = T^{-1}$. Además, el que la inversa exista es condición necesaria y suficiente para que la transformación sea unitaria. El lema siguiente predice que esto no va a ocurrir en la mayoría de los casos.

Lema 4.2 *Sea T una transformación lineal definida a partir de un esquema de orden $o(H)$ sobre un genotipo de longitud L, tal que*

- *sus primeras $L - o(H)$ columnas se forman dejando los bits fijos del esquema con sus valores originales, y siendo las demás posiciones igual a cero, salvo la posición correspondiente al i-ésimo "$*$", que es igual a 1;*
- *y siendo el resto de las columnas las mismas de la matriz identidad en esas posiciones,*

entonces T es unitaria si y sólo si no hay "0"s ni más de un "1" en las primeras $L - o(H)$ posiciones, $o(H_{0..L-o(H)}) \leq 1$.

Dem:

Para la construcción dada, $\det(T) = det(A_{L-o(H)})$, donde $A_{L-o(H)}$ es el menor principal de orden $L - o(H)$, por lo que podemos determinar la invertibilidad de T a partir de la invertibilidad de $A_{L-o(H)}$.

Respecto de $A_{L-o(H)}$, vemos que por construcción:

- *Un 0 del esquema se transforma en una fila de "0"s.*
- *Un 1 del esquema se transforma en una fila de "1"s.*
- *Cada uno de los primeros $L - o(H)$ "$*$" del esquema genera una componente distinta de la base canónica.*

Cualquier "0" añade una fila completa de ceros al menor principal, y dos "1"s o más generan dos o más vectores idénticos, y por lo tanto no independientes linealmente. Estas son las dos situaciones que impiden que el menor sea invertible.

En cualquiera de los otros dos casos, las componentes de la base canónica son linealmente independientes entre sí y, en caso de haber un "1", con el vector de "1"s.

□

Corolario 4.3 *Sea una transformación lineal T como la definida en el lema anterior. T es unitaria si y sólo si hay a lo sumo un bit fijo en las primeras $L - o(H)$ posiciones del esquema y, en caso de haberlo, es un "1".*

La probabilidad –promediada sobre todos los problemas– de que T sea invertible se puede aproximar (suponiendo que $p(0) = p(1) = 1/2$, lo que sabemos que no es verdad, ya que hay en general bastantes más "0"s que "1"s en las soluciones) por

$$\begin{aligned} P(T_{\text{invertible}}) &= P(\text{ninguna posición fija}) + P(\text{un 1 fijo}) \\ &\simeq 2^{-L+o(H)} + (L - o(H))\, 2^{-L+o(H)-1} \end{aligned} \tag{4.7}$$

Podemos ver el resultado para el caso $L = 50$ en la Fig. 4.8.

Hemos planteado este resultado negativo para mostrar por un lado qué queremos lograr, y por el otro, con qué problemas nos encontramos a la hora de buscar una transformación unitaria. En el apartado siguiente proponemos una solución para este inconveniente.

4.3.3. Matriz directa del análisis de permutación de la identidad

Nos podemos fijar en que hay $o(H)$ columnas de relleno tomadas de la matriz identidad y que, según la demostración del Lema 4.2, cada una de estas columnas contiene un único "1", que a la hora de calcular el determinante cancela el efecto de la fila donde se encuentra. El problema de invertibilidad viene dado por las $o(H)$ posiciones fijas, por lo que si en lugar de colocar los

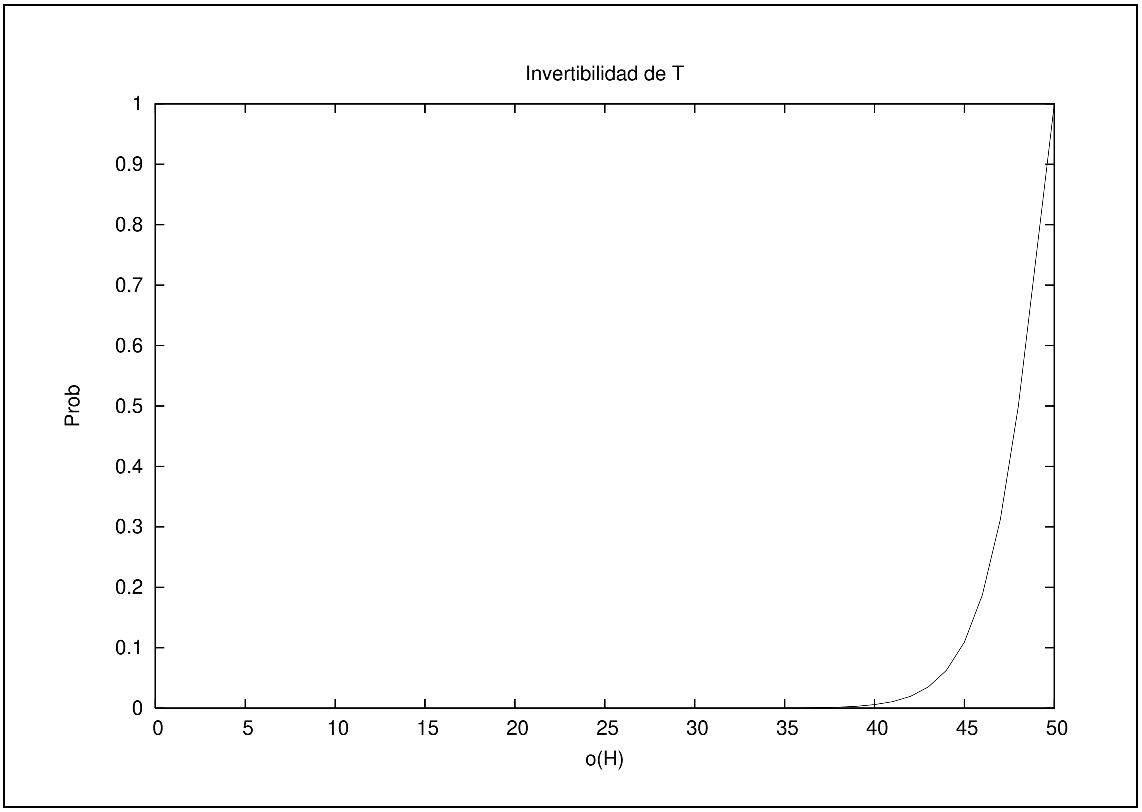


Figura 4.8: Probabilidad de que T sea invertible

"1"s de las columnas de relleno según la matriz identidad, lo hacemos según las posiciones fijas del esquema, T siempre es invertible, con independencia de su orden.

Por ejemplo,

$$T = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

El inconveniente de esta matriz es, sin embargo, que ya no supone una perturbación mínima de la matriz identidad, sino una permutación de ésta, y una perturbación posterior.

4.3.4. Matriz de una permutación del esquema con perturbación mínima de la identidad

Con la misma idea del apartado anterior, vamos a permutar el esquema, en lugar de la matriz identidad. Con esto conseguimos dos objetivos:

- La permutación es una inversión que agrupará los bits significativos, lo que probablemente quiere decir que existe un cierto grado de epistasis entre ellos.
- La perturbación posterior es una transformación lineal que actúa sólo sobre los vectores que hemos seleccionado.

Definición 4.4 (Permutación del esquema) *Sea H un esquema; se define una permutación P tal que el esquema permutado $H_P = P \cdot H$ tiene la forma $H_P = [* \dots * \, d \dots d]$, donde d son las posiciones fijas del esquema.*

La permutación consiste en llevar todos los "$*$" al principio del esquema, y los bits fijos al final. Sobre el esquema del ejemplo anterior tendríamos

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Nótese que una permutación P sobre el genotipo, es equivalente a un reetiquetado del grafo, $P^T G P$, operación que no cambia su topología.

Podemos ahora generar la matriz de la TL como en el apartado 4.3.2, sabiendo que T siempre es invertible, por lo que T es una TLU.

$$\begin{aligned} T &= \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \\ &= \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ M & \mathbb{I} \end{bmatrix} \end{aligned} \tag{4.10}$$

Con la partición por bloques de la matriz T, podemos comprobar que se trata de una matriz diagonal inferior. En trabajos anteriores [45] se han empleado este tipo de matrices para hacer compresión del espectro. Estas matrices tienen además una propiedad interesante, y es que son raíces segundas de la matriz identidad.

Lema 4.5 *Sea T una transformación lineal sobre un esquema con la permutación de la Def. 4.4 según el Lema 4.2. Entonces siempre existe la inversa de T, y $T^{-1} = T$.*
Dem:
Operando,

$$\begin{aligned} T \cdot T &= \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ M & \mathbb{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ M & \mathbb{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbb{I} \end{bmatrix} = \mathbb{I} \end{aligned} \tag{4.11}$$

□

Podemos resumir el cambio de espacios que producen T y P en la Fig. 4.9.

4.3.5. Generación del esquema a partir de las estadísticas de la población

En los apartados anteriores se ha descrito cómo contruir la matriz de la transformación lineal a partir de un esquema de referencia. A continuación explicaremos cómo generar el esquema de referencia a partir de las estadísticas de la población.

En un primer análisis el algoritmo corre sin ninguna transformación, acumulando estadísticas sobre la frecuencia con la que aparecen "1"s en cada posición; cuando se cumple un criterio de parada determinado, estas cantidades acumuladas se emplean para asignar a cada posición un valor en el rango $[-1{,}0\ 1{,}0]$, de modo que:

- El módulo del valor indica el grado de determinismo de la posición; así, para módulos próximos a 1,0, la posición tomará casi siempre el mismo valor, mientras que para módulos próximos a 0,0, la posición tomará valores de "0" o "1" indistintamente.

- El signo del valor indica la tendencia, a valor "0" $(-)$ o a valor "1" $(+)$.

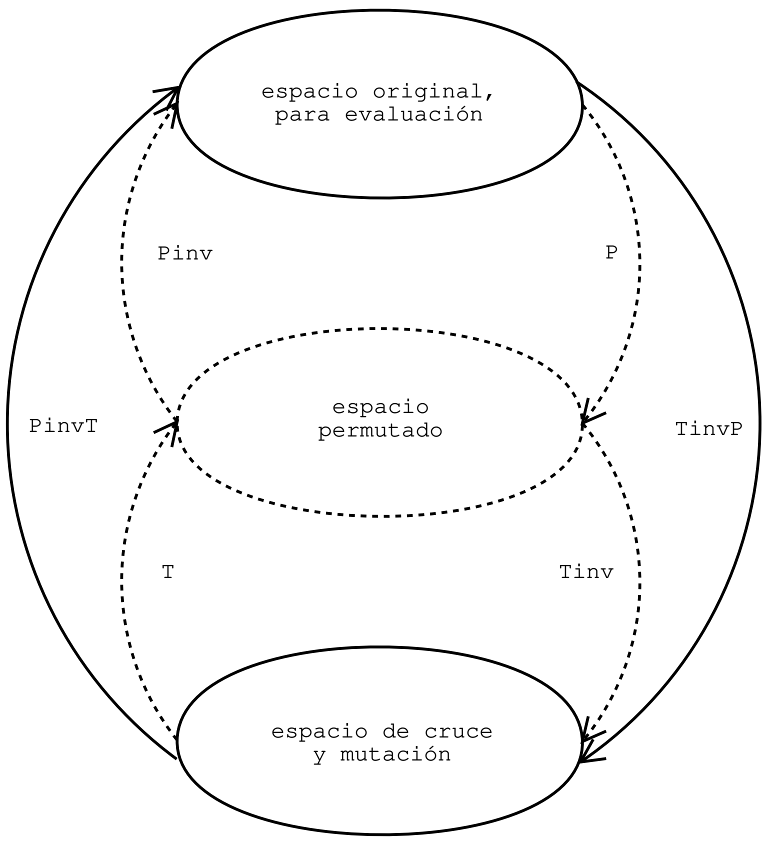


Figura 4.9: Transformaciones T y P

Seguidamente se realiza un ordenamiento de las posiciones en función de su grado de determinismo; existen dos posibilidades para definir la función que ordena las posiciones:

- Dando prioridad a los valores de módulo más alto.

- Ordenado primero los valores positivos (correspondientes a los "1"), y después los negativos (correspondientes a los "0").

Estas opciones se expondrán desde un punto de vista de la implementación en el Cap. 5, y la comparación de sus prestaciones se verá en el Cap. 6. La razón de estudiar estas dos posibilidades radica en el hecho, ya comentado, de que los "1" del genotipo tienen más información que los "0", por lo que parece más lógico darles mayor protección. Pero, al mismo tiempo, no sabemos si para el algoritmo será más importante la protección de las posiciones más deterministas, tengan el valor que tengan.

Por último, se determina un número de posiciones fijas, bien directamente como un porcentaje sobre la longitud del genotipo, o bien a partir de un valor umbral relacionado con el grado de determinismo de las posiciones.

Capítulo 5

Software

5.1. El programa gadnh

`gadnh` toma como entrada el nombre de un fichero de texto que describe un problema de Steiner en un grafo, según el formato de la biblioteca OR [5], y lo resuelve por medio de un AGH (Algoritmo Genético Híbrido), cuyo algoritmo local es el HRD (Heurístico de Red de Distancias).

El programa puede generar una población inicial, correr el AGH sobre ella, realizar un análisis de esquemas a partir del cual crea la matriz de una transformación lineal, recuperar el estado del generador de números pseudoaleatorios y volver a correr el AGH, esta vez transformando a los individuos para la evaluación.

5.1.1. Requisitos no funcionales

El programa se ha compilando a partir de código C++ con g++ 2.95.4 20011002 (Debian prerelease), y se ha probado con éxito en procesadores i686.

Debido a la implementación del analizador sintáctico en la biblioteca EO, son necesarios 352 M de RAM y 300 M de Swap para compilarlo con la optimización `-O -O2`. Es posible reescribir el código perteneciente a esta parte con variables intermedias para el analizador sintáctico, lo que facilita la compilación, pero lo hace menos legible.

El programa ocupa 3,1 M en el disco. La cantidad de memoria consumida y el tiempo de ejecución variará dependiendo del problema que se analice. Para el problema `steinb14.txt` (100 vértices, 25 de ellos seleccionados), con una población de 100 individuos y análisis de esquemas, puede rondar los 2 M y los 25 min en un PIII a 866 Mhz.

5.1.2. Requisitos funcionales

`gadnh` ofrece una serie de opciones que permiten modificar el tipo de análisis y resolución del problema. A continuación puede verse una captura de la ayuda del programa, que se puede invocar mediante `./gandh -h` o bien `./gandh --help`.

```
bin/gadnh:

Usage: bin/gadnh [Options]
Options of the form "-f[=Value]" or "--Name[=value]"
Where:

######    General    ######
-h, --help:     Prints this message
            Optional. By default: 0
--stopOnUnknownParam:   Stop if unkonwn param entered
            Optional. By default: 1
-r, --seed:     random number seed
            Optional. By default: 1046025554
--maria:        kisses for María
            Optional. By default: 0
-F, --file:     problem file
            Required. By default:

######    GA    ######
-p, --popsize:  population size
            Optional. By default: 100
--schemagen:    schemas will be searched for these generations
                (0 to disable, -1 non stop)
            Optional. By default: 0
--statstype:    0: generations average, 1: only last generation
            Optional. By default: 0
--matrixtype:   0: determ, 1: half unit
            Optional. By default: 0
--sorttype:     0: ones and zeros, 1: only ones
            Optional. By default: 0
--schemapercent:        this fraction [0.0, 1.0] of genotype bits
                        will be fixed for schemas
            Optional. By default: 0
--schemathres:  bits with randomness < thres [0.0, 1.0]
                will be fixed for schemas
```

```
            Optional. By default: 0

######    GA: Crossover and Mutation    ######
-x, --crosspoints:      crossover points
            Optional. By default: 1
-c, --crossrate:        crossover rate
            Optional. By default: 0.9
-b, --bitfliprate:      probability that each bit is flipped
            Optional. By default: 0.01
-m, --mutarate: individual's mutation rate
            Optional. By default: 1

######    GA: Selection    ######
-s, --selection:        0: deterministic tournament, 1: ranking
            Optional. By default: 0
-R, --replacement (only for schema):    0: generational, 1: elitism
            Optional. By default: 0
--toursize:     deterministic tournament size
            Optional. By default: 10
--rankpress:    ranking pressure [1.0, 2.0]
            Optional. By default: 2
--rankexp:      ranking exponent (1.0 for linear ranking)
            Optional. By default: 1

######    GA: Stop    ######
-G, --maxgen:   maximum number of generations
            Optional. By default: 1000
-g, --mingen:   minimum number of generations
            Optional. By default: 10
--steadygen:    number of generations without improvement to stop
            Optional. By default: 20
-O, --enoughopt:        if != 0, value good enough to stop both runs
            Optional. By default: 0

######    Persistence    ######
-S, --status:   status file to save parameters to
            Optional. By default: bin/gadnh.status
-M, --monitor:  a file to save statistics to
            Optional. By default: .stats
-C, --checkgen: generation interval to save state (0 to disable)
            Optional. By default: 0
```

```
-T, --checktime:        time interval (sec.) to save state (0 to disable)
            Optional. By default: 0
--gchk_filename:        a file name root to save state to, by generation
            Optional. By default: generation
--tchk_filename:        a file name root to save state to, by time
            Optional. By default: time
-L, --load:     a state file to restart from (output files will be
                overwritten)
            Optional. By default:

@param_file      defines a file where the parameters are stored
```

Damos una explicación más amplia de las opciones del programa, así como de su implicación en el análisis:

`-h, --help` Muestra la ayuda.

`--stopOnUnknownParam` Control de errores de entrada. Si el analizador sintáctico no puede procesar algún parámetro de entrada, detiene el programa.

`-r, --seed` Permite fijar la semilla del generador de números pseudoaleatorios. Si el parámetro no se introduce manualmente, se genera aleatoriamente.

`--maria` Muestra el mensaje "n `kisses for María`", donde n es el valor del parámetro.

`-F, --file` Es el único parámetro obligatorio, e indica el nombre del fichero de texto con formato de la biblioteca OR que describe el problema que se va a analizar.

`-p, --popsize` Tamaño de la población en cada generación. Aumentarla incrementa notablemente el tiempo de ejecución, pero si es demasiado baja, se puede no tener la suficiente diversidad genética. En las pruebas de laboratorio se ha trabajado con una población de `100` individuos.

`--schemagen` Número de generaciones sobre la que se realiza el análisis de esquemas. `--enoughopt` fuerza su valor a -1, lo que supone un análisis continuo hasta que el algoritmo cumpla algún otro criterio de parada. Esta combinación es la que se ha empleado en los resultados finales de laboratorio, ya que permite no tener un conocimiento previo sobre el tiempo que tarda el algoritmo en converger.

`--statstype` Tipo de estadísticas que se recogen de los esquemas. Con valor `0` se analizan los individuos de cada generación, desde la primera a la última, mientras que con valor `1` sólo se analizan los de la última.

`--matrixtype` Tipo de transformación lineal que emplea el algoritmo. En este Proyecto se ha empleado el tipo `1`, como aparece descrito en la sección 4.3. El tipo `0` se desechó en la fase de análisis previo, pero se ha mantenido por motivos históricos.

`--sorttype` Al hacer el análisis de esquemas, la aleatoriedad del valor de los genes va a asociada a un valor que permite reordenar estos. Esta opción permite seleccionar qué tipo de orden se sigue; para el valor `0` tienen prioridad los más deterministas, independientemente de si tienden a valor “0” o “1” en el genotipo. Para el valor `1`, los “1” del genotipo tienen prioridad sobre los “0”. En este Proyecto se han estudiado ambas posibilidades.

`--schemapercent` Una de las dos formas de indicar qué porción del esquema se van a considerar bits fijos. Se redondea al valor entero superior más cercano del porcentaje deseado, sobre la longitud total del genotipo.

`--schemathres` La segunda forma de indicar qué porción del esquema se van a considerar bits fijos. Al hacer el análisis de los esquemas, a cada posición se le asigna un valor real entre `0.0` y `1.0` que indica el grado de determinismo de esa posición, además de un signo, que indica si la posición tiende a “0” ($-$) o a “1” ($+$). Se consideran posiciones fijas del esquema todas aquellas cuyo valor real en módulo sea mayor o igual que el del umbral introducido con esta opción.

`-x, --crosspoints` Número de puntos para el operador de cruce. En las simulaciones el valor utilizado fue `2`. La biblioteca EO permite modificar fácilmente el código para introducir tipos de cruce nuevos, como el uniforme.

`-c, --crossrate` Probabilidad de que dos padres seleccionados se crucen. En las pruebas se utilizó la tasa más alta, `1.0`.

`-b, --bitfliprate` Probabilidad de que cada bit invierta su valor, para los individuos que mutan. En las pruebas se fijó un valor de `0.01`.

`-m, --mutarate` Probabilidad de que cada individuo mute. La opción `-b` y ésta se podrían haber combinado en una sola, pero así es más sencillo modificar el código para añadir otros tipos de mutación que no son de inversión de bit.

-s, --selection Tipo de selección que realiza el AGH. En este Proyecto se hicieron las pruebas con el valor `0`, que corresponde al torneo determinista. Los resultados previos con el valor `1` (clasificación) fueron bastante negativos, por lo que se abandonó esa línea de trabajo. La biblioteca EO hace sencillo modificar el código para introducir tipos de selección nuevos.

-R, --replacement Tipo de reemplazamiento que realiza el AGH. En este Proyecto se ha hecho el análisis para el valor `0` (generacional) y valor `1` (con elitismo).

--toursize En caso de que la selección sea por torneo determinista, esta opción indica el número de individuos que compiten en el mismo. En este Proyecto se usó un valor de `5`.

--rankpress En caso de que la selección sea por clasificación, esta opción selecciona el valor de presión, según vimos en el apartado 3.3.7. Las pruebas previas se hicieron con un valor de `2.0`.

--rankexp En caso de que la selección sea por clasificación, esta opción selecciona el valor del exponente, según vimos en el apartado 3.3.7. Las pruebas previas se hicieron con un valor de `1.0`, correspondiente a la clasificación lineal.

-G, --maxgen Criterio de parada que indica el número máximo de generaciones por análisis que puede correr el algoritmo. Es una medida de seguridad para evitar que en caso de pérdida total de variedad genética, por estancamiento en un mínimo local, el algoritmo no se detenga y retrase el resto de las simulaciones. En este Proyecto se hizo una estimación previa del tiempo de convergencia necesario para cada problema, y después se fijó este parámetro multiplicando ese tiempo por un factor 15.

-g, --mingen Criterio de parada que indica, en caso de usar además la opción `--steadygen`, el número mínimo de generaciones por análisis que debe correr el algoritmo.

--steadygen Criterio de parada que indica el número de generaciones sin mejora del valor de evaluación que se le permiten al algoritmo antes de tener que detenerse. En este Proyecto se hicieron los análisis previos con un valor de `30`, pero para las simulaciones se usó la opción `--enoughopt`.

-O, --enoughopt Criterio de parada que indica al algoritmo que debe parar cada análisis al alcanzar un determinado valor de evaluación. Este procedimiento es muy común en problemas de optimización, donde se puede tener una idea del valor del óptimo, pero no se sabe cómo lograrlo. En este Proyecto, dado que se conoce el valor exacto del óptimo, se permitió un empeoramiento de aproximadamente el 2 % (valor `0.02`).

`-S, --status` La biblioteca EO permite crear un fichero con los valores de las opciones empleados en la última simulación, de forma que las llamadas sucesivas se pueden hacer por referencia a este fichero, sin necesidad de introducir todos los valores manualmente. Se prefirió, sin embargo, la potencia y flexibilidad de los guiones de `bash` para lanzar las simulaciones.

`-M, --monitor` Los resultados del algoritmo se pueden guardar en un fichero de estadísticas. En las simulaciones, los nombres de los ficheros siguieron la plantilla *parámetros_problema_porcentaje_iteración*`.stats`, donde

- *parámetros* describe el uso o no de elitismo, y la forma de generar los esquemas: `elite`, `noelite`, `eliteones` y `noeliteones`;
- *problema* es el nombre del fichero con los datos del problema, p.ej. `steinb14.txt`;
- *porcentaje* es la fracción de bits del esquema que se consideran fijos, p.ej. `0.70`;
- *iteración* es el número de repetición para un problema dado, con unos parámetros fijos, p.ej. 100 iteraciones entre `0` y `99`.

`-C, --checkgen` La biblioteca EO permite guardar el estado de las simulaciones cada cierto número de generaciones. Esta opción está desactivada por defecto, y no es incompatible con la opción `-T, --checktime`.

`-T, --checktime` La biblioteca EO permite guardar el estado de las simulaciones cada cierto tiempo, fijado por el valor dado en segundo. Esta opción está desactivada por defecto, y no es incompatible con la opción `-C, --checkgen`.

`--gchk_filename` En caso de activar la opción `-C, --checkgen`, se puede especificar la raíz del nombre de los ficheros donde se salva el estado de la simulación.

`--tchk_filename` En caso de activar la opción `-T, --checktime`, se puede especificar la raíz del nombre de los ficheros donde se salva el estado de la simulación.

`-L, --load` En caso de disponer de ficheros de los generados por las opciones `-C, --checkgen` o `-T, --checktime` esta opción permite especificar el nombre del fichero donde está guardado el estado a partir del cual se quiere reiniciar la simulación. Hay que tener cuidado al usarla porque los ficheros de resultados se sobreescriben.

5.1.3. Ejemplo de un fichero de resultados

El programa `gadnh` produce unos ficheros de estadísticas con un aspecto parecido al de, por ejemplo, `noelite_steinb10.txt_0.50_63.stats` (generado en la fase de análisis previos), que a continuación vamos a analizar por partes. En primer lugar, aparece una cabecera con los datos de los parámetros del problema.

```
#this file:             results/noelite_steinb10.txt_0.50_63.stats
#holds stats for:       data/steinb10.txt
#
#rand seed: 1045193874
#selection type: deterministic tournament: 5 individuals
#replacement type: generational
#population size: 100
#genotype length (fixed by graph problem): 62
#crossover points: 2
#crossover rate: 1
#probability of bit-flip mutation: 0.01
#mutation rate: 1
#maximum number of generations: 400
#runs will stop if they reach value: 88
#schema analysis:
#       generations: -1
#       fixed bits: 31
#
#Stats:
#Gen.   Eval.   Best    Average & Stdev
```

Dependiendo de los valores de las opciones de la línea de órdenes, puede haber o no líneas que pertenezcan a determinados objetos (en este caso son de `eoSchemaStat`).

```
# eoSchemaStat: matrix type set to: minimum perturbation
#               of identity, always invertible
```

```
# eoSchemaStat: stats type set to: average of the
#                                  last generation
```

A continuación comienza a correr el algoritmo por primera vez, en caso de haber seleccionado la obtención de estadísticas para usar una transformación lineal. Las columnas significan, respectivamente,

- número de generación;
- número acumulado de evaluaciones;
- mejor valor de evaluación en la población;
- valor medio de evaluación en la población y
- desviación típica de los valores de evaluación en la población.

Estas líneas no empiezan por `#`, para que las puedan interpretar programas como `gnuplot`.

```
1       200     155     195.69 14.1533
2       300     155     181.88 12.2022
3       400     138     169.91 12.981
4       500     135     156.03 11.9439
5       600     126     145.78 8.44062
6       700     118     138.06 8.46242
7       800     112     131.07 7.51061
8       900     108     126.13 7.90174
9       1000    106     119.36 7.84125
10      1100    98      113.92 6.56849
11      1200    98      108.68 4.71164
12      1300    94      105.13 4.83183
13      1400    94      102.58 5.08946
14      1500    92      100.28 4.19495
15      1600    90      98.75 4.53355
16      1700    87      97.48 5.40198
```

Si se ha seleccionado el análisis de esquemas, hay una salida con el esquema que generará la transformación lineal (en este caso sólo hay "0" y "$*$", pero en general habrá también "1"), tanto antes como después de la permutación. Además se indica explícitamente qué permutación se ha usado.

```
# eoSchemaStat::getStats: number of ones in schema is 0

# schema (rows of up to 10 elements):
# * 0 * * * 0 * 0 * 0
# * 0 * 0 0 * 0 0 * *
# * 0 * 0 0 * 0 0 * 0
# * 0 * 0 0 0 * * 0 *
# * 0 0 0 * * 0 * 0 0
# * * * 0 0 0 * * 0 *
# 0 *
# permuted schema (rows of up to 10 elements):
# * * * * * * * * * *
# * * * * * * * * * *
# * * * * * * * * * *
# * 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0

# permutation (rows of up to 10 elements):
# 0 2 3 4 6 8 10 12 15 18
# 19 20 22 25 28 30 32 36 37 39
# 40 44 45 47 50 51 52 56 57 59
# 61 1 5 7 9 11 13 14 16 17
# 21 23 24 26 27 29 31 33 34 35
# 38 41 42 43 46 48 49 53 54 55
# 58 60
```

A continuación corre el algoritmo por segunda vez; en esta ocasión, los individuos sufren una transformación lineal.

```
1       1900    143     194.97 13.791
2       2000    147     180.51 13.6059
3       2100    141     165.23 11.8235
4       2200    133     154.22 9.82841
5       2300    128     145.42 7.84854
6       2400    120     138.84 7.02912
7       2500    111     132.15 8.64727
8       2600    103     124.75 8.09586
9       2700    105     118.15 6.71554
10      2800    100     113.88 5.28057
```

```
11      2900    97      109.38 5.18502
12      3000    96      105.34 5.2382
13      3100    96      103.29 6.18616
14      3200    95      99.25 3.80954
15      3300    94      98.67 3.44965
16      3400    93      98.48 4.64166
17      3500    93      97.35 4.61306
18      3600    93      97.71 5.62695
19      3700    92      95.8 3.91062
20      3800    92      95.16 3.7382
21      3900    91      95.67 4.84936
22      4000    91      94.89 4.59885
23      4100    91      94.56 4.69756
24      4200    90      94.31 4.84235
25      4300    90      94.33 5.44218
26      4400    90      94.18 4.5157
27      4500    86      93.23 4.79658
```

Por último, cuando el algoritmo se detiene, se muestra el genotipo del mejor individuo, con su valor de evaluación (`86`), su longitud (`62`), y el tiempo total que ha necesitado la simulación.

```
#best individual:
#86  62 0000100000000100001000100000000
#       0000011000000000000001000010001
#time: 93 sec(1.55 min)
```

5.2. Clases

Toda la estructura de herencia y relación de clases de la biblioteca EO se encuentra en la documentación de la misma [47]. Presentamos aquí sólo las clases creadas a propósito para este Proyecto. Las flechas con trazo continuo indican la estructura de herencia; las de trazo discontinuo, un puntero a un método de otra clase; las de trazo punteado, su relación con otras clases en el programa.

Dado que el programa `gadnh` está pensado para investigación, con la idea de dejarlo abierto para futuros cambios (no estrictamente *mejoras*), no describimos los métodos de la clase de la forma usual, sólo con el valor de retorno y la salida de los parámetros por referencia. En los casos que nos ha parecido apropiado, además se indica de forma esquemática el modo de funcionamiento general del método.

5.2.1. eoSchemaStat $<$ EOT $>$

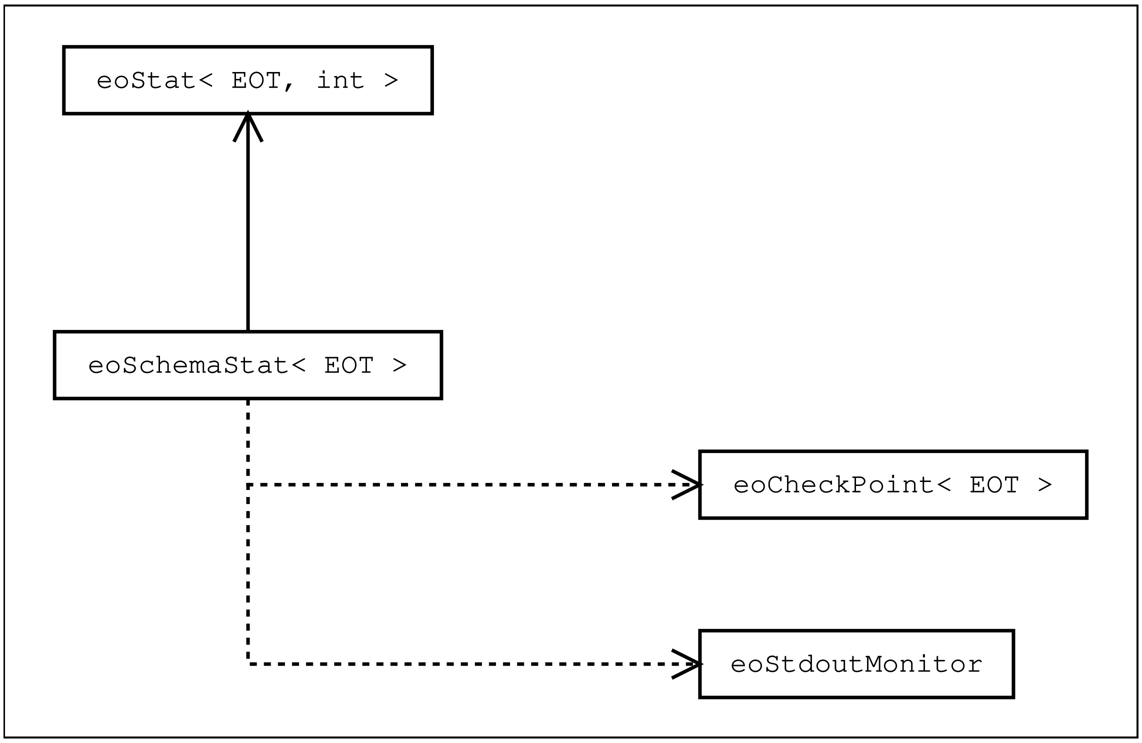


Figura 5.1: Jerarquía de la clase eoSchemaStat $<$ EOT $>$

`#include include/eoSchemaStat.hpp`

Tipos

```
enum MatrixType{MAT_DETERM, MAT_HALFI, NMAT};
enum StatsType{STATS_AVG, STATS_END, NSTATS};
enum SortType{SORT_ONESZEROS, SORT_ONES, NSORT};
enum SchemaElement{seZERO, seWILDCARD, seONE};
typedef std::vector<SchemaElement> Schema;
typedef std::vector<unsigned> Permutation;
```

En `MatrixType` hay un elemento por cada tipo de matriz implementada (`NMAT` es el número de elementos). `MAT_DETERM` es una matriz creada como un $\mathbb{I} \oplus M$, donde $\mathbb{I}$ es la matriz identidad, y M es una plantilla del esquema, cuyo uso se desestimó en los análisis previos (se ha mantenido por motivos históricos, pero no se ha documentado). `MAT_HALFI` es la matriz usada en este Proyecto para la transformación lineal, según la descripción del apartado 4.3.4.

En `StatsType` hay un elemento por cada tipo de estadísticas implementadas. `STATS_AVG` corresponde a la toma de estadísticas como media de todos los individuos y todas las generaciones, mientras que `STATS_END` corresponde a la toma de estadísticas sólo en la última generación.

En `SortType` hay un elemento por cada método de permutación de los bits del genotipo. `SORT_ONESZEROS` corresponde a dar prioridad a los más deterministas, tiendan a "0" o a "1", mientras que `SORT_ONES` corresponde a dar prioridad a los "1" sobre los "0", y dentro de estos grupos, a los más determinisitas sobre los más aleatorios.

En `SchemaElement` se encuentran los elementos de un esquema, que pueden ser "0" (`seZERO`), "*" (`seWILDCARD`) o "1" (`seONE`).

`Schema` es un vector (`vector<>` de la STL) cuyos elementos son `SchemaElement`. Una vez terminada la adquisición de estadísticas, guarda el esquema a partir del cual se genera la transformación lineal.

`Permutation` es un vector (`vector<>` de la STL) que guarda la permutación realizada sobre el esquema, en caso de que la haya.

Métodos

```
eoSchemaStat(std::string _description = ``Sch. eval gens'',
             unsigned int _len = 0);
eoSchemaStat(char _type = MAT_DETERM,
             int _value = -1,
             std::string _description = ``Sch. eval gens'',
             unsigned int _len = 0);
eoSchemaStat(int _value = -1,
             std::string _desc = ``Sch. eval gens'',
             unsigned int _len = 0);
```

eoSchemaStat(): Constructor.

```
~eoSchemaStat();
```

~eoSchemaStat(): Destructor.

```
void operator()(const eoPop<EOT>& _pop);
```

operador(): Si el objeto se añade a un `eoCheckPoint< EOT >`, en cada generación se le pasará a este operador la población de individuos.

```
void triggerLastEval(void);
```

triggerLastEval(): Detiene la adquisición de estadísticas y crea la transformación lineal inmediatamente. Útil cuando el criterio de parada depende de un objeto externo, y el programa necesita indicar de algún modo que se evalúen ya las estadísticas.

```
static bool cmp_abspairs(const std::pair<unsigned int, float> &x,
                         const std::pair<unsigned int, float> &y);
```

cmp_abspairs(): Función de comparación (para `sort()` de la STL) que ordena los elementos del esquema según su grado de determinismo, sin importar si tienden a "0" o "1".

```
static bool cmp_onesfirstpairs(
                  const std::pair<unsigned int, float> &x,
                  const std::pair<unsigned int, float> &y);
```

cmp_pairs(): Función de comparación (para `sort()` de la STL) que ordena los elementos del esquema primero los que tienden a "1" antes que los que tienden a "0", y dentro de estos, según su grado de determinismo.

```
unsigned int size(void);
```

size(): Devuelve la longitud del genotipo.

```
bool done(void);
```

done(): Indica si ya se han calculado las estadísticas.

```
void setOrder(unsigned int _order);
```

setOrder(): Fijar manualmente el número de bits fijos del esquema.

```
void setThreshold(float _thres);
```

setThreshold(): Fijar el umbral de aleatoriedad permitido a los bits fijos del esquema. Todos los bits con un valor de determinismo igual o superior a éste se condierarán bits bijos del esquema.

```
void setMatrixType(char _type);
```

setMatrixType: Fijar el tipo de matriz que se empleará para la transformación lineal.

```
void setStatsType(char _type);
```

setStatsType(): Fijar el tipo de estadísticas que se van a tomar (a lo largo de todas las generaciones, o sólo en la última).

```
void setSortType(char _type);
```

setSortType: Fijar cómo se van a ordenar los bits para decidir su prioridad a la hora de fijarlos en el esquema (si los "1" tienen prioridad sobre los "0" o sólo importa el nivel de aleatoriedad).

```
static unsigned int getStats(Schema& _v);
static unsigned int getStats(Schema& _v,
                             Permutation& _p);
```

getStats(): Devuelve 0 si todavía se están tomando estadísticas. Si no, asigna a cada posición un valor $y_i = 2(\bar{x}_i - 0{,}5)$, donde $\bar{x}_i$ es el número medio de "1" en esa posición, en función del tipo de estadísticas seleccionado. Después define una permutación sobre el esquema de forma que queden al principio los bits más deterministas, en función del tipo de ordenamiento de bits seleccionado. Dependiendo del orden fijado, o del nivel de umbral, un número determinado de bits, contando desde el principio del genotipo permutado, se traduce en bits fijos del esquema, y el resto quedan como "$*$". Si no hay errores, devuelve el número de bits fijados como valor de retorno, y por referencia un vector donde los `true` son posiciones fijas del esquema, y los `false` posiciones "$*$".

```
static bool lin_trans(const EOT& _indi,
                      EOT& _indi_trans);
```

Si la transformación lineal no está definida, y todavía no están listas las estadísticas, aplica la transformación identidad a `_indi` y devuelve `false`. Si las estadísticas están listas, pero la matriz todavía no se ha generado, la crea. Si la matriz ya se ha generado, transforma el individuo con `matvecprod()`.

```
static void matvecprod(const EOT& _indi,
                       bool **_a,
                       EOT& _indi_trans);
```

lin_ trans(): Aplica la transformación lineal a `_indi` para generar `_indi_trans`.

```
void setFilename(const string& _filename);
```

setFilename(): Opcionalmente, los mensajes de salida del objeto se pueden mandar a un fichero, además de a la salida estándar. Se usa para añadir estos mensajes al fichero de estadísticas `.stats`.

```
static void makeMAT_HALFI(bool **_pa,
                          const Schema& _v,
                          const Permutation& _p);
```

makeMAT_HALFI: Genera la matriz de la transformación lineal según el método descrito en el apartado 4.3.4.

5.2.2. eoEvalFuncWithGraph < EOT >

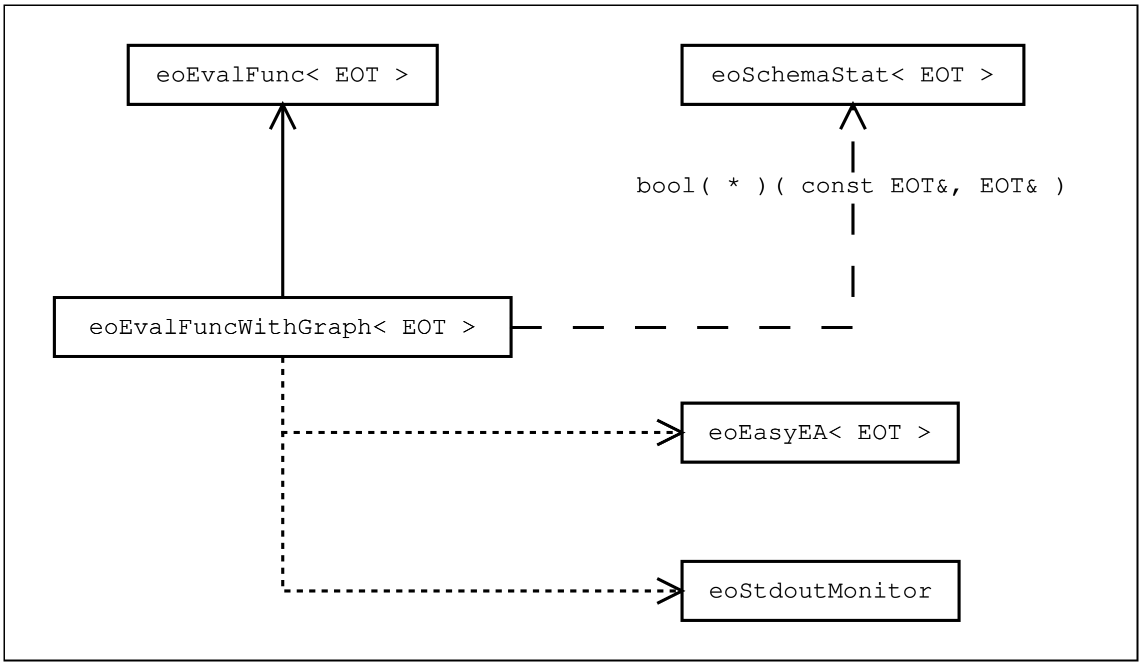


Figura 5.2: Jerarquía de la clase eoEvalFuncWithGraph < EOT >

```
#include include/eoEvalFuncWithGraph.hpp
```

Métodos

```
eoEvalFuncWithGraph(SteinerGraph& _g);
```

eoEvalFuncWithGraph(): Constructor.

```
~eoEvalFuncWithGraph();
```

~eoEvalFuncWithGraph(): Destructor.

```
void operator()(EOT & _indi);
```

operator(): Evalúa un individuo, si no fue evaluado ya en generaciones anteriores. Si el objeto tiene algún puntero a una función de transformación (lineal o no lineal), llama a la transformación antes de pasarlo como entrada del HRD.

```
void addTransform(bool (*_transform)(const EOT&, EOT&));
```

addTransform(): Captura un puntero a una función de transformación de los genotipos. La función puede hacer cualquier tipo de transformación, pero debe cumplir las siguientes especificaciones:

1. Devolver `true` si el individuo se ha transformado, y `false` en caso contrario.

2. Si el individuo no se ha transformado, se debe aplicar la transformación identidad.

5.3. Implementación del HRD

La implementación del Heurístico de Red de Distancias se encuentra en los ficheros `dnh.cpp` y `include/dnh.hpp` (véase el apartado 5.4.1). El código se escribió principalmente sobre la biblioteca BGL (Boost Graph Library) [48].

De forma paralela a la descripción funcional del HRD en la sección 4.2, presentamos ahora la descripción detallada de la implementación del heurístico.

5.3.1. Cálculo del grafo de distancias $D(G)$

Una forma eficiente de calcular y almacenar tanto las distancias como los caminos más cortos entre todos los pares de vértices del grafo es buscar, para cada vértice, el árbol de distancia mínima a cada uno de los otros vértices, y usar una matriz de distancias y una matriz de predecesores para guardarlo.

El elemento d_{ij} La matriz de distancias representa la distancia mínima entre los vértices i y j, mientras que el elemento p_{ij} de la matriz de predecesores, representa cuál es el vértice anterior a j para llegar a i en el árbol generado a partir de i.

Para el cálculo de las distancias se empleó la función `dijkstra_shortest_paths()` de la BGL aplicada a cada vértice. Esta función está escrita con cola de prioridad. El cálculo se realiza una sola vez en cada análisis del algoritmo genético híbrido. La complejidad de esta función es $\mathcal{O}(|E|\log n)$

5.3.2. Construcción del subgrafo G_1 de $D(G)$ inducido por W

Esta parte del algoritmo aprovecha el cálculo del apartado anterior.

5.3.3. Cálculo de un árbol generador mínimo T_1 de G_1

Se emplea el algoritmo de Prim, al ser G_1 un grafo completo, implementado en la función `prim_minimum_spanning_tree()` de la BGL, cuya complejidad es $\mathcal{O}(|E|\log n)$.

5.3.4. Expansión de T_1 en G_2

La generación del grafo nuevo se hace a partir de la matriz de predecesores del primer apartado.

5.3.5. Cálculo de un árbol generador mínimo T_2 de G_2

En este caso el grafo no es completo, en general, por lo que se emplea el algoritmo de Kruskal, implementado en la función `kruskal_minimum_spanning_tree()`, con complejidad $\mathcal{O}(|E|\log|E|)$.

5.3.6. Poda de las ramas sobrantes

La poda se realiza de forma iterativa con la función `prune_one_edge()`, descrita en el apartado 5.4.5.

5.4. Árbol de desarrollo

A continuación damos una pequeña explicación de la función de cada fichero, salvo los guiones de `bash` (ficheros `.sh`), que analizaremos en la sección siguiente. El árbol de directorios y ficheros se puede ver en la Fig. 5.3.

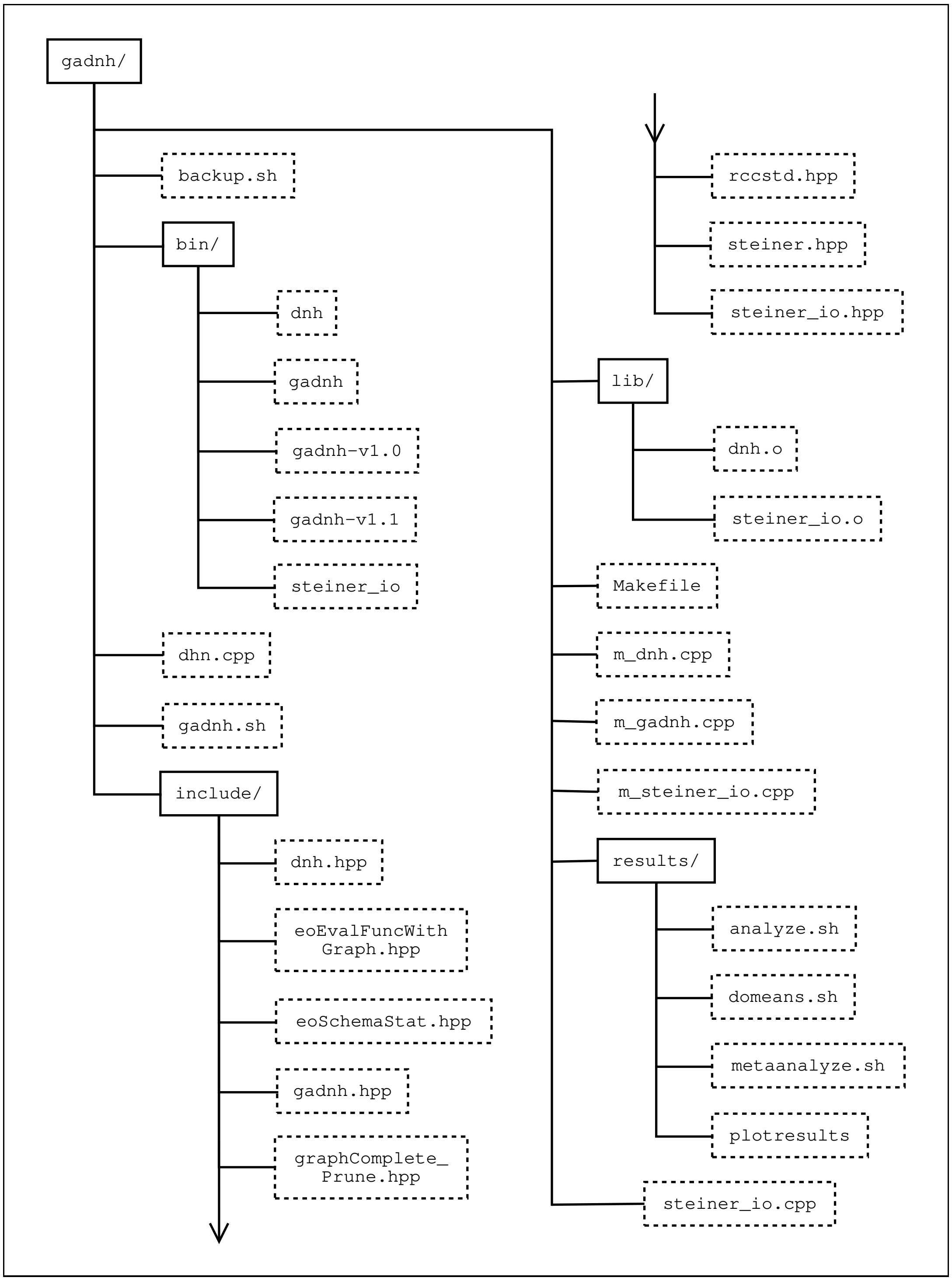


Figura 5.3: Árbol de desarrollo del software del Proyecto.

5.4.1. dnh.cpp e include/dnh.hpp

Contienen el código del HRD, que se presenta como la función `dnh()`.

```
int dnh(SteinerGraph& g,
        std::vector<std::vector<int> >& d_array,
        std::vector<std::vector<steiner::vertex_descriptor> >&
        p_array);
int dnh(SteinerGraph& g,
        std::vector<std::vector<int> >& d_array,
        std::vector<std::vector<steiner::vertex_descriptor> >& p_array,
        bool SHOWSOLTREE)
```

Las matrices de distancia y predecesores se pueden dar como parámetros de entrada, con lo que se evita su cálculo en la función. En caso de que no se den, la función calculará y devolverá estos valores por referencia.

5.4.2. include/eoEvalFuncWithGraph.hpp

Contiene la declaración y definición de la clase eoEvalFuncWithGraph<EOT>, ya descrita en la sección 5.2.

5.4.3. include/eoSchemaStat.hpp

Contiene la declaración y definición de la clase eoSchemaStat<EOT>, ya descrita en la sección 5.2.

5.4.4. include/gadnh.hpp

Contiene la definición de los individuos. La notación de la biblioteca EO puede inducir a error, ya que queremos minimizar el valor de evaluación, no el de adaptación.

```
typedef eoBit<eoMinimizingFitness> Indi;
```

Además encontramos dos sobrecargas del operador de salida, `operator<<`,

```
template <class T>
std::ostream& operator<<(std::ostream& os,
                         std::vector<T>const & v);
```

para imprimir cómodamente vectores de tipos sencillos, y

```
template <class First, class Second>
std::ostream& operator<<(std::ostream& os,
                         std::vector<
                                     std::pair<First, Second>
                                    >const & v);
```

para imprimir los vectores de genotipos reordenados según el grado de aleatoriedad de sus posiciones.

5.4.5. include/graphComplete_Prune.hpp

Contiene las funciones que permiten completar un grafo y podarlo.

```
template <class G>
int prune_one_edge(G& g);
```

Busca alguna rama del grafo que se pueda podar, situación que se da cuando una rama no termina en un vértice seleccionado. En caso de que exista, poda una arista y devuelve `1`. Si no, devuelve `0`.

```
template <class G>
int makecompletegraph(G& g,
                      std::vector<std::vector<int> >& d_array,
                      std::vector<std::vector<
                                              steiner::vertex_descriptor>
                                             >& p_array);
```

A partir de la matriz de distancias y de predecesores, sutituye el grafo G por el grafo de distancias $D(G)$.

5.4.6. include/rccstd.hpp

Es una cabecera de propósito general, que define una macro para depurar programas

```
#define PUF(x) std::cout << "Puf(" << x << ")!" << endl;
```

y algunos códigos de error

```
#define EX_MEM "error assigning memory"
#define EX_FNOTOPEN "file cannot be opened"
#define EX_FNOTSAVED "file cannot be saved"
```

5.4.7. include/steiner.hpp

Contiene las declaraciones de los tipos de datos que representan los grafos y sus propiedades, según se emplean en los programas de este Proyecto. Las declaraciones principales se exponen a continuación.

```
enum vertex_to_be_spanned_t {
  vertex_to_be_spanned
};
BOOST_INSTALL_PROPERTY(vertex, to_be_spanned);
```

Crea una propiedad nueva para los vértices de la biblioteca Boost Graph.

```
typedef std::pair<int,int> Edge;
typedef int WeightType;
typedef property <vertex_index_t,
                  int,
                  property <vertex_to_be_spanned_t, bool> > VertexProperty;
typedef property <edge_weight_t,
                  WeightType,
                  property< edge_weight2_t, int > > EdgeProperty;
typedef adjacency_list <listS,
                        vecS,
                        undirectedS,
                        VertexProperty,
                        EdgeProperty > SteinerGraph;
```

Define el tipo de las aristas, las propiedades de vértices y aristas, y la representación de los grafos (`adjacency_list`).

5.4.8. steiner_io.cpp e include/steiner_io.hpp

Contienen las declaraciones y el código de varias funciones de entrada / salida para los grafos del Proyecto.

```
int readsteinerORfile(steiner::SteinerGraph& g, char *s);
```

Permite leer a memoria el grafo descrito por un fichero con el formato de la biblioteca OR.

```
void skipspaces(std::ifstream& fp, char& c);
```

Una forma cómoda de ignorar los espacios en blanco de un fichero de texto.

```
int readchar2int(std::ifstream& fp, char& c);
```

Lee en un `int` un número descrito por una cadena de caracteres.

```
int writesteinerDOTfile(steiner::SteinerGraph& g, char *s);
```

Permite escribir un grafo a un fichero con un formato apropiado para el programa `dotty` de representación automática de grafos.

5.4.9. `Makefile`

Es el fichero que permite compilar el programa por partes, y en conjunto, de forma cómoda, mediante la orden `make` *`parámetros`* `...`.

5.4.10. `m_dnh.cpp` e `m_steiner_io.cpp`

Son dos programas de control para comprobar el funcionamiento de las funciones implementadas en `dnh.?pp` y `steiner_io.?pp`, respectivamente.

5.4.11. `m_gadnh.cpp`

A diferencia de los anteriores, no es un guión de control, sino el fichero donde se encuentra el `main()` del programa `gadnh`.

5.5. Guiones de `bash`

Los guiones de `bash` son programas sencillos que permiten combinar otros programas para que actúen sobre una gran cantidad de ficheros, lo que los hace muy potentes. Muchas veces, a la hora de escribir un guión, no se persigue el grado de optimización o elegancia que es habitual en un programa escrito en un lenguaje como C++, sino llegar a una solución suficiente en el menor tiempo posible.

En este Proyecto se han escrito y utilizado varios guiones; para hacer las copias de seguridad, lanzar las simulaciones de forma fácil y distribuida, y para analizar los ficheros de resultados.

5.5.1. `backup.sh`

Este pequeño guión crea un directorio en `/tmp/` donde guarda los ficheros más importantes del Proyecto. Se seleccionan aquellos que no se pueden volver a descargar de otro sitio, o recompilar, para buscar un compromiso entre el tiempo necesario para volver al trabajo tras un fallo en el sistema y el espacio necesario. Luego este directorio se comprime con `tar` y se sube a un servidor del laboratorio.

Toma como parámetro de entrada una cadena de texto entre comillas dobles, que guardará en un fichero dentro del directorio de la copia de seguridad, de forma que se pueda identificar fácilmente. Al terminar, borra los datos temporales.

5.5.2. `gadnh.sh`

Aunque la biblioteca EO proporciona los medios (por medio de los ficheros de parámetros) para poder relanzar una simulación desde un fichero de texto, este método está un poco limitado si se quieren hacer muchas iteraciones sobre varios problemas.

El guión `gadnh.sh` tiene una serie de variables al inicio con las que se puede definir qué problemas se van a simular, con qué porcentajes de bits fijos en los esquemas y cuántas iteraciones por cada porcentaje, en qué directorio se van a guardar los resultados, y qué tipo de reemplazamiento y reordenamiento del esquema se va a llevar a cabo.

Sólo aparecen estas variables porque eran las mínimas necesarias para realizar los estudios de laboratorio deseados.

5.5.3. `results/analyze.sh` y `metaanalyze.sh`

`analyze.sh` toma un fichero de estadísticas como entrada, y analiza sus líneas para obtener los datos interesantes para las gráficas.

La salida es una línea de texto con estos datos separados por tabulaciones:

1. Un resumen de algunos parámetros de la simulación, separados por "/".
2. El valor del óptimo para este problema.
3. El mejor valor de evaluación obtenido en la última generación del análisis de esquemas del algoritmo.
4. La generación en la que se detuvo el análisis de esquemas.
5. El mejor valor de evaluación obtenido en la última generación del algoritmo.
6. La generación en la que se detuvo el algoritmo.

7. La fracción que representan realmente los bits fijos del esquema respecto de la longitud total.

8. El valor de la semilla para el generador de número pseudoaleatorios usada.

`metaanalyze` corre `analyze` sobre una serie de ficheros, y agrupa las líneas que éste crea (sólo con los valores 2 a 7 de la relación anterior) en un único fichero.

5.5.4. `results/domeans.sh` y `plotresults.sh`

`domeans` lee el fichero creado por `metaanalyze`, y usa `octave` para buscar los valores medios.

`plotresults` toma el fichero generado por `domeans` y por medio de `gnuplot` muestra en pantalla una gráfica donde se puede ver la mejora o emperamiento en velocidad y calidad del algoritmo con la trasnformación lineal respecto del algoritmo sin la transformación lineal.

5.6. La biblioteca PARI

La inversión de matrices binarias es un problema en sí mismo, especialmente con el software científico del que se dispone habitualmente (como `octave`). Una aproximación válida para matrices pequeñas es

$$A_2^{-1} = \mathrm{mod2}\left(\mathrm{redondeo}\left(A_{10}^{-1} \cdot |A_{10}|\right)\right) \tag{5.1}$$

donde A_2 y A_{10} son la matriz a invertir, en base 2 y base 10, respectivamente, $\mathrm{mod2}(\cdot)$ es la función módulo 2, y $|\cdot|$ es el determinante.

Sin embargo, para tamaños moderados de la matriz, como 20x20, la falta de precisión de los tipos de datos internos hacen perder la información del bit menos significativo, precisamente la única que requiere la función $\mathrm{mod2}(\cdot)$, por lo que la aproximación no es válida.

La solución consiste o bien en usar lenguajes que soporten precisión arbitraria, o bien con un tipo de datos específico para operaciones en módulo 2. La biblioteca libre PARI en C, y el programa asociado GP [51] ofrecen esto y muchas más posibilidades relacionadas con la teoría de números.

Finalmente, el desarrollo de las transformaciones de este Proyecto llevó a matrices para las que, en caso de existir la inversa, ésta era igual a la matriz original, por lo que no fue necesario emplearla.

En el momento de escribir esta memoria, un gazapo no resuelto de la biblioteca no permite usarla cómodamente como sistema secundario de cálculo, ya que si se inicializa la pila que usa internamente, se trabaja con ella, y se libera la memoria, no se puede volver a inicializar.

Parte II

Resultados de laboratorio

Capítulo 6

Resultados experimentales

6.1. Descripción de las simulaciones

Para poder evaluar la validez de nuestro método, debemos buscar resultados estadísticamente representativos, ya que la teoría del espectro, las transformaciones lineales, y su relación con las prestaciones del AG todavía no están lo suficientemente desarrolladas como para respaldar un estudio por si solas.

Como vimos en el apartado 3.6.5, incluso si esta teoría fuera sólida, tampoco están desarrolladas todavía las herramientas para estimar el espectro de una forma fiable.

Por lo tanto, se seleccionarán una serie de parámetros y, para unos determinados valores, se hará que el algoritmo corra repetidamente con distintas inicializaciones. Así podremos extraer resultados estadísticos significativos.

6.1.1. Problemas sobre los que se trabaja

Los problemas de Steiner de la biblioteca OR están agrupados en las categorías B, C, D y E. Las dimensiones de los grafos que contienen varían según la relación siguiente:

Grafos B de 50 a 100 vértices, y de 63 a 200 aristas.

Grafos C de 500 vértices, y de 625 a 12.500 aristas.

Grafos D de 1.000 vértices, y de 1.250 a 25.000 aristas.

Grafos E de 2.500 vértices, y de 3.125 a 62.500 aristas.

Problema	$t_{1\ \mathbf{sim}}$	$t_{1000\ \mathbf{sim}}$
`steinb01.txt`	18,6 s	5,2 h
`steinb02.txt`	22,0 s	6,1 h
`steinb03.txt`	31,7 s	8,8 h
`steinb04.txt`	18,7 s	5,2 h
`steinb05.txt`	21,4 s	5,9 h
`steinb06.txt`	34,7 s	9,6 h
`steinb07.txt`	73,8 s	20,5 h
`steinb08.txt`	82,3 s	22,9 h
`steinb09.txt`	109,0 s	30,3 h
`steinb10.txt`	77,3 s	21,5 h
`steinb11.txt`	97,3 s	1,1 d
`steinb12.txt`	107,0 s	1,2 d
`steinb13.txt`	440,9 s	5,1 d
`steinb14.txt`	549,1 s	6,3 d
`steinb15.txt`	493,7 s	5,7 d
`steinb16.txt`	380,3 s	4,4 d
`steinb17.txt`	432,7 s	5,0 d
`steinb18.txt`	521,0 s	6,0 d

Cuadro 6.1: Estimación de los tiempos de ejecución.

6.1.2. Limitación por tiempos de ejecución

En primer lugar se hizo un estudio del tiempo aproximado de simulación, que limitaría la investigación en función de los equipos de los que se dispone en el laboratorio. En este estadio basta una estimación gruesa, por lo que se corrieron 10 iteraciones[1] por problema del grupo B para calcular el tiempo medio que requiere cada uno. En la columna $t_{1\ \mathrm{sim}}$ del Cuadro 6.1 están recogidos estos tiempos.

Para poder hablar de mejoras o empeoramientos del orden del 5 %, son necesarias unas 100 simulaciones. Se estimó que por cada problema con unos parámetros dados se estudiarían 10 valores distintos, lo que resulta en 1.000 simulaciones por problema, dados unos parámetros. Esto aumenta el tiempo de simulación tres órdenes de magnitud, lo que se encuentra reflejado en la columna $t_{1000\ \mathrm{sim}}$ del Cuadro 6.1.

Los tiempos de ejecución previstos para el grupo B llegan hasta los 6 días por problema. Otros factores a tener en cuenta son:

[1] El equipo utilizado fue un PIII 866 Mhz 352 M SDRAM

- Las simulaciones se pueden correr en paralelo en varios equipos, que son además más potentes que el equipo de prueba. Esto puede bajar los tiempos en un factor 50.
- Se quieren analizar varios problemas, cada uno con varios parámetros, lo que aumenta los tiempos en un factor 20.
- Las pruebas se hicieron sin el segundo análisis, en el que se emplea la transformación lineal. El tiempo de cálculo de ésta es despreciable, así como el tiempo de transformación de los individuos que introduce, por lo que en el análisis real podemos esperar tiempos aumentados por un factor 2, por este motivo.
- Las dimensiones de los grafos de los grupos están al menos un orden de magnitud por encima de las del grupo B, y esperamos una complejidad mínima del algoritmo de $\mathcal{O}(mn^2)$ debida al HRD (incluso si despreciamos el efecto del aumento del tamaño de los genotipos o el número de individuos por generación), lo que puede suponer un factor prohibitivo de 3 órdenes de magnitud o mayor.

Teniendo en cuenta estos factores, se decidió restringir el estudio al grupo B, y buscar unos criterios de selección de problemas dentro de este grupo, que exponemos a continuación.

6.1.3. Limitación por tiempo de convergencia

La segunda restricción que encontramos para los problemas seleccionables consiste en el tiempo de convergencia del algoritmo. Para poder ver mejoras, el análisis no puede acabar en un número demasiado pequeño de generaciones.

Esta condición, junto con la del apartado anterior, nos indica que los candidatos al análisis son problemas con muchas generaciones, pero que cada de ellas se calcule rápidamente. El tiempo necesario para cada generación depende a su vez de las características del grafo y de la longitud del genotipo. Las generaciones para las que el HRD hace el cálculo en poco tiempo corresponden a grafos simples que no tienen demasiado interés.

Fue necesario buscar un compromiso entre el interés del problema y el tiempo de simulación requerido. La estimación de las generaciones necesarias para la convergencia se hizo a partir de los mismos resultados del apartado anterior. En el Cuadro 6.2 se presentan los resultados.

También se analizó el resultado que es capaz de conseguir el HRD por sí solo (que es el mismo valor que consigue el individuo cuyo genotipo es una cadena de “0”) y la pérdida de calidad que supone respecto del óptimo, lo que se puede observar en el Cuadro 6.3.

Problema	**Gen.**
`steinb01.txt`	9,5
`steinb02.txt`	9,0
`steinb03.txt`	3,5
`steinb04.txt`	11,0
`steinb05.txt`	7,6
`steinb06.txt`	6,3
`steinb07.txt`	20,0
`steinb08.txt`	19,0
`steinb09.txt`	8,6
`steinb10.txt`	21,8
`steinb11.txt`	30,8
`steinb12.txt`	6,8
`steinb13.txt`	43,3
`steinb14.txt`	41,3
`steinb15.txt`	14,1
`steinb16.txt`	32,3
`steinb17.txt`	24,3
`steinb18.txt`	14,7

Cuadro 6.2: Estimación de los tiempos de convergencia sin la TL.

Problema	**Óptimo**	**HRD**	**Pérdida (%)**
`steinb01.txt`	82	82	0,0
`steinb02.txt`	83	90	8,4
`steinb03.txt`	138	140	1,4
`steinb04.txt`	59	64	8,5
`steinb05.txt`	61	62	1,6
`steinb06.txt`	122	128	4,9
`steinb07.txt`	111	111	0,0
`steinb08.txt`	104	104	0,0
`steinb09.txt`	220	223	1,4
`steinb10.txt`	86	98	14,0
`steinb11.txt`	88	90	2,3
`steinb12.txt`	174	174	0,0
`steinb13.txt`	165	175	6,1
`steinb14.txt`	235	238	1,3
`steinb15.txt`	318	325	2,2
`steinb16.txt`	127	137	7,9
`steinb17.txt`	131	134	2,3
`steinb18.txt`	218	226	3,7

Cuadro 6.3: Valor del árbol de Steiner mínimo y resultado del HRD.

6.1.4. Selección de problemas y parámetros del análisis

A partir de los resultados de los dos apartados anteriores, se decidió analizar en la primera fase los problemas `steinb10.txt` y `steinb11.txt`. Ambos problemas son un compromiso entre tiempo de ejecución y de convergencia; además, para el primero el HRD consigue un resultado un 14 % peor que el óptimo, mientras que para el segundo, la degradación de calidad es parecida a la que permitimos al algoritmo para detenerse. Esto nos permitirá la comparación de las prestaciones del método en los dos casos.

Los parámetros fijos del análisis son[2]:

`--statstype=1` Análisis de esquemas en la última generación.

`--matrixtype=1` La matriz descrita en el apartado 4.3.4.

`--crosspoints=2` Cruce por dos puntos.

`--crossrate=1.0` Tasa de cruce del 100 % para maximizar su efecto.

`--bitfliprate=0.01` Tasa de inversión de bit del 1 %.

`--selection=0` Selección por torneo determinista.

`--toursize=5` Torneo entre 5 individuos.

Los parámetros variables son:

`--enoughopt` El algoritmo se detiene cuando alcanza un valor de evaluación un 2 % peor que el óptimo. Los valores de los óptimos se encuentran en el Cuadro 6.3.

`-G` El número máximo de generaciones permitidas al algoritmo por cada análisis, como medida de seguridad. El valor se calculó a partir del Cuadro 6.2, con un factor 25.

`-R` Se analizaron los casos con elitismo y sin elitismo.

`--sorttype` Se analizaron los casos de ordenamiento dando preferencia a los "1" del esquema, y sin dársela.

[2] Se puede encontrar una descripción más amplia de las opciones en el apartado 5.1.2

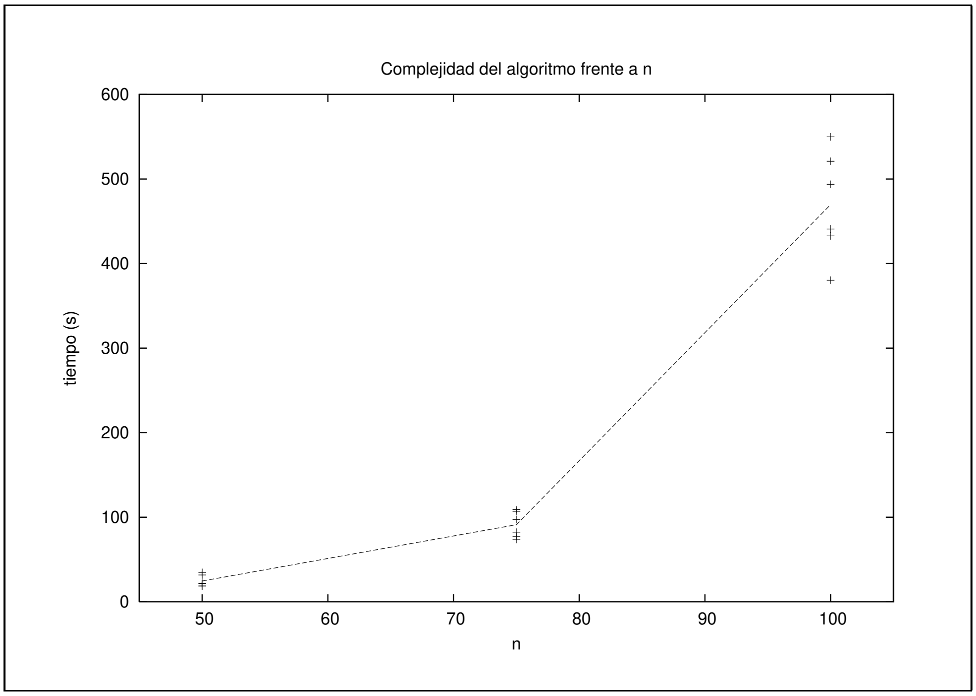


Figura 6.1: Complejidad, número de vértices.

6.2. Complejidad temporal del algoritmo

Se comprobó con los datos disponibles la complejidad temporal del algoritmo respecto a los parámetros de los grafos. La relación se hizo con los tiempos del Cuadro 6.1 frente al número de vértices, n (Fig. 6.1), el número de vértices seleccionados, m (Fig. 6.2), el número de aristas, $|E|$ (Fig. 6.3), la longitud del genotipo, $n - m$ (Fig. 6.4), la complejidad del HRD, $\mathcal{O}(mn^2)$ (Fig. 6.5), y la aproximación a la complejidad del algoritmo que da Esbensen [14], $\mathcal{O}(m^2)$ (Fig. 6.6).

Salvo para la Fig. 6.1, donde parece haber cierta ley en el crecimiento, en los demás casos los resultados no son los suficientemente uniformes como para sacar alguna conclusión. Consideramos que la muestra no es lo suficientemente amplia para hacer este estudio de complejidad.

6.3. Efecto en el tiempo de convergencia de la TLU

Para estudiar el efecto de la TLU en la convergencia de los problemas `steinb10.txt` y `steinb11.txt`, se fijaron los parámetros de trabajo según

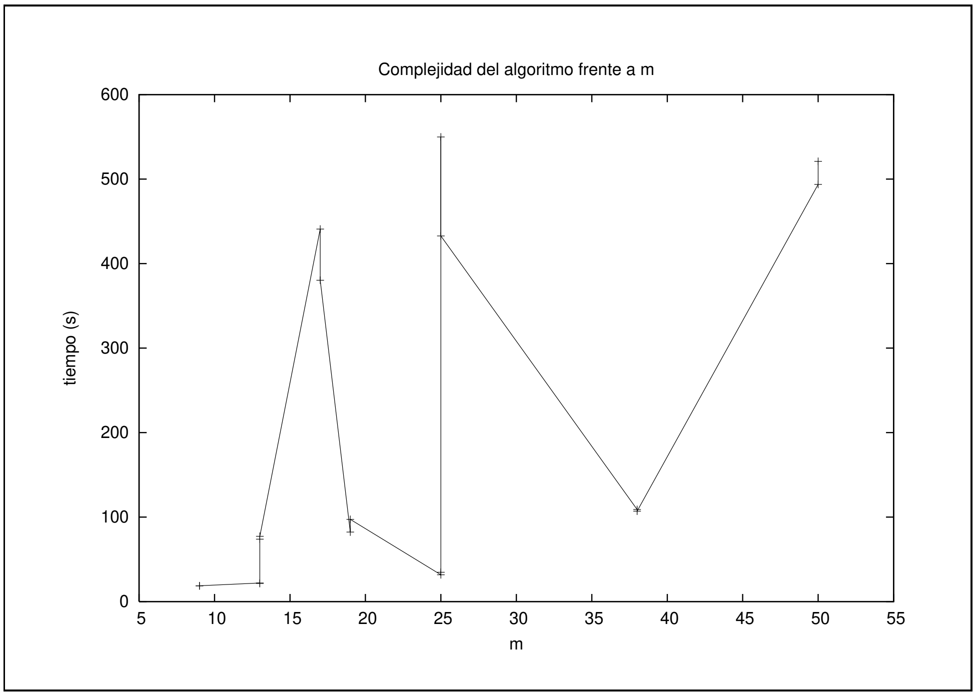


Figura 6.2: Complejidad, número de vértices seleccionados.

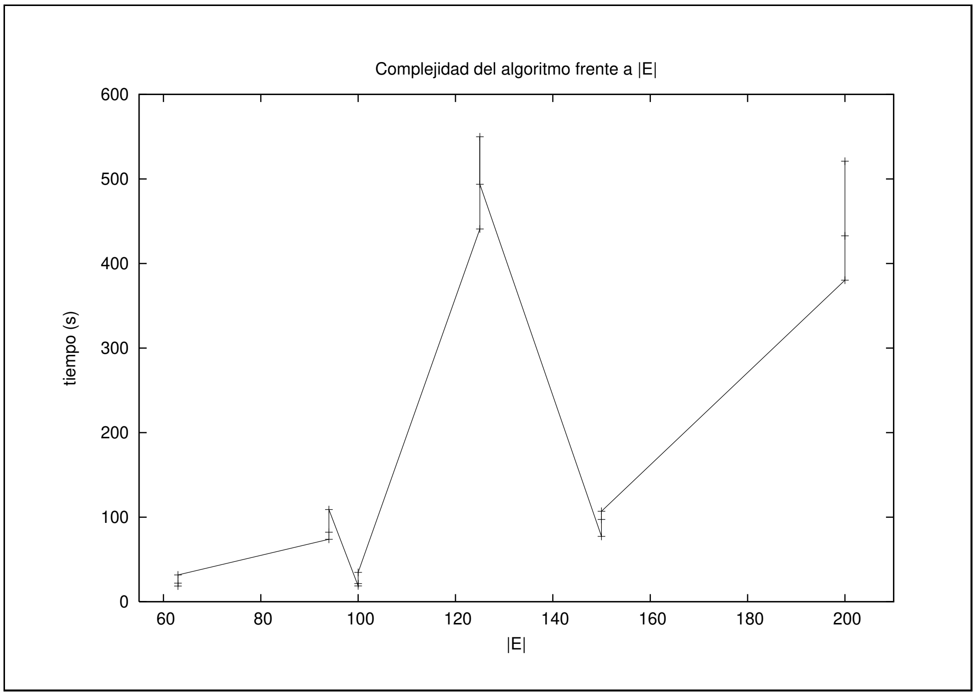


Figura 6.3: Complejidad, número de aristas.

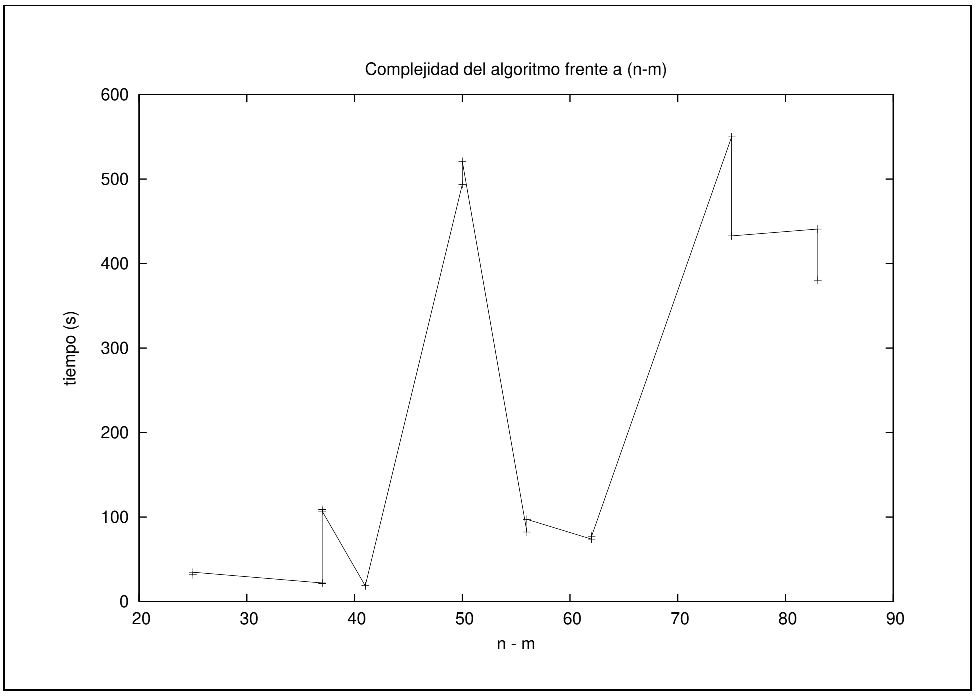


Figura 6.4: Complejidad, longitud del genotipo.

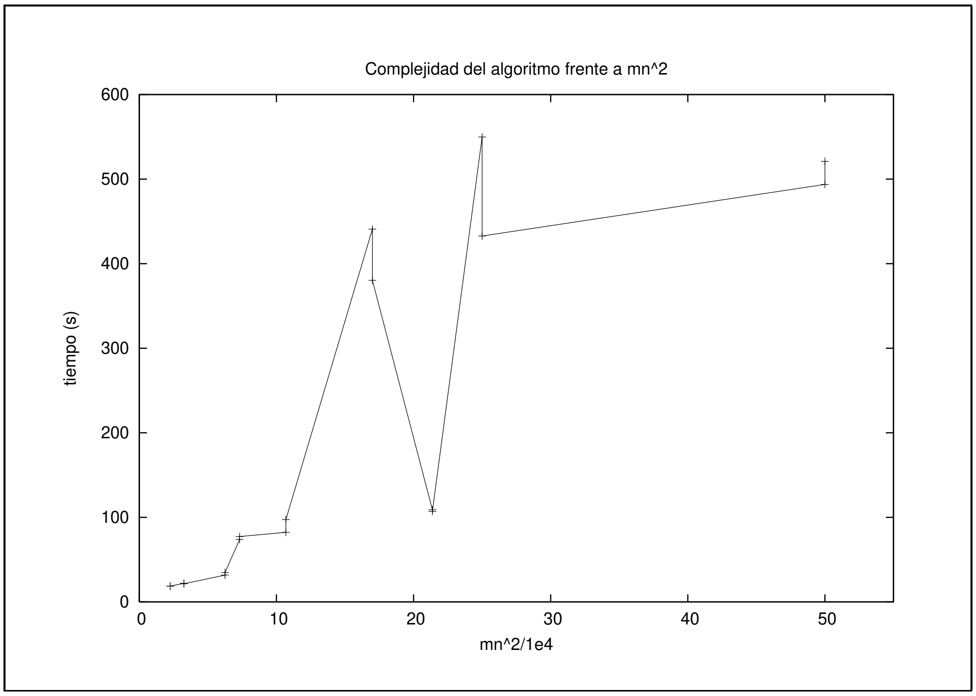


Figura 6.5: Complejidad, $\mathcal{O}(mn^2)$.

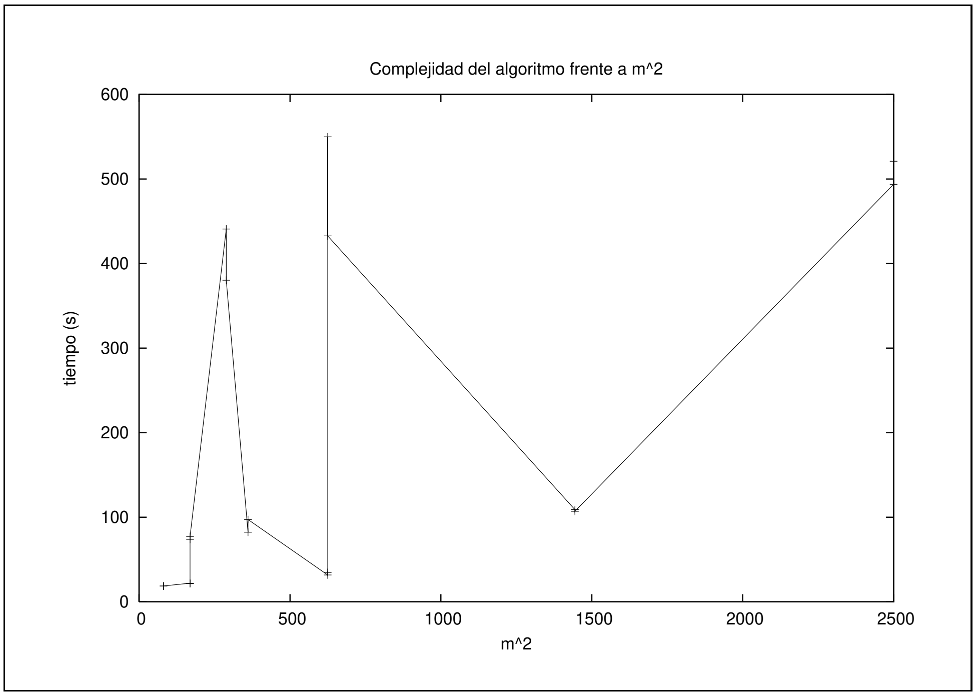


Figura 6.6: Complejidad, $\mathcal{O}(m^2)$.

la descrición del apartado 6.1.4.

El interés se centra en la mejora o empeoramiento que producen el elitismo (opción `-R`) y el método de ordenamiento de las estadísticas para crear el esquema (opción `--sorttype`). En la discusión que sigue, se utilizará la terminología:

`E=0` Para las simulaciones sin elitismo.

`E=1` Para las simulaciones con elitismo.

`S=0` Si el ordenamiento de las estadísticas no da prioridad a los “1” sobre los “0”.

`S=1` Si el ordenamiento de las estadísticas sí da prioridad a los “1” sobre los “0”.

6.3.1. Caso `E=0 S=1`

Se pueden apreciar en Fig. 6.7 y Fig. 6.8 unos resultados muy negativos. En ambos casos, el tiempo de convergencia aumenta entre un 150 % y un 200 %, que es justamente el efecto contrario al que buscamos.

6.3.2. Caso `E=1 S=1`

Tratamos de separar el efecto del elitismo del de la prioridad de los "1" en el ordenamiento de las estadísticas, activando la primera opción. La Fig. 6.9 y la Fig. 6.10 siguen mostrando resultados negativos, pero también se puede comprobar una mejora sobre el caso anterior. Si suponemos que el efecto de ambos parámetros es separable, podemos considerar que el elitismo introduce una mejora en el comportamiento del algoritmo, por lo que a partir de lo que vimos en el apartado 3.6.3, la deriva genética sería un problema mayor que la pérdida de diversidad.

6.3.3. Caso `E=1 S=0`

Para comprobar las conclusiones del apartado anterior, se mantuvo el elitismo y se quitó la prioridad de los "1" en el ordenamiento de las estadísticas. Los resultados de este apartado se encuentran en las Fig. 6.11 y Fig. 6.12.

En estos dos casos, el método está funcionando correctamente. Para esquemas con un orden en torno a dos valores, 30 % y 70 % de la longitud del genotipo, la mejora en la velocidad de convergencia se sitúa entre el 20 % y el 25 %.

Es significativo que las gráficas tengan un aspecto simétrico; en la construcción de la matriz hay un toma y daca entre el orden y la cantidad de los esquemas que se protegen. Este resultado puede indicar que el método funciona correctamente en esos dos puntos de compriso, aunque hará falta extender los experimentos a otros problemas en la fase segunda del análisis.

6.3.4. Caso `E=0 S=0`

Para terminar la primera fase del análisis, se comprobó si efectivamente el elitismo tiene un efecto positivo en nuestro algoritmo, independientemente del ordenamiento de estadísticas. Los resultados pueden verse en la Fig. 6.13 y Fig. 6.14.

6.3.5. Resumen de resultados experimentales de la primera fase

Se ha podido establecer de forma estadísticamente significativa, sobre los problemas `steinb10.txt` y `steinb11.txt`, que el método de selección de esquemas para la transformación lineal que da prioridad a los "1" sobre los "0" descrito en el apartado 4.3.5, empeora dramáticamente las prestaciones del algoritmo, por lo que debe ser desechado.

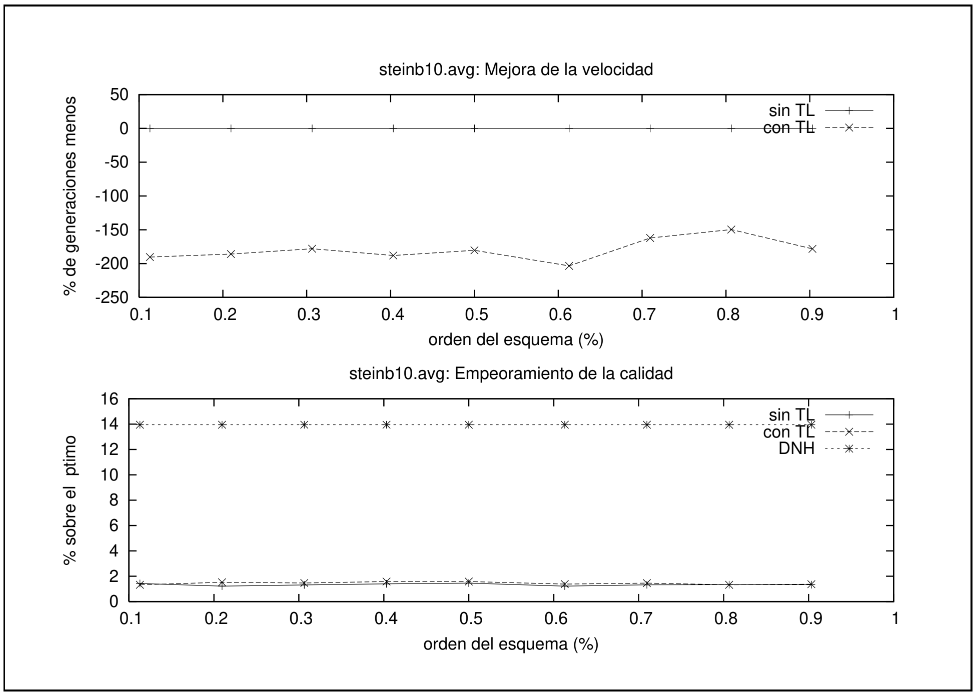


Figura 6.7: Resultados de convergencia para `steinb10.txt` con `E=0 S=1`.

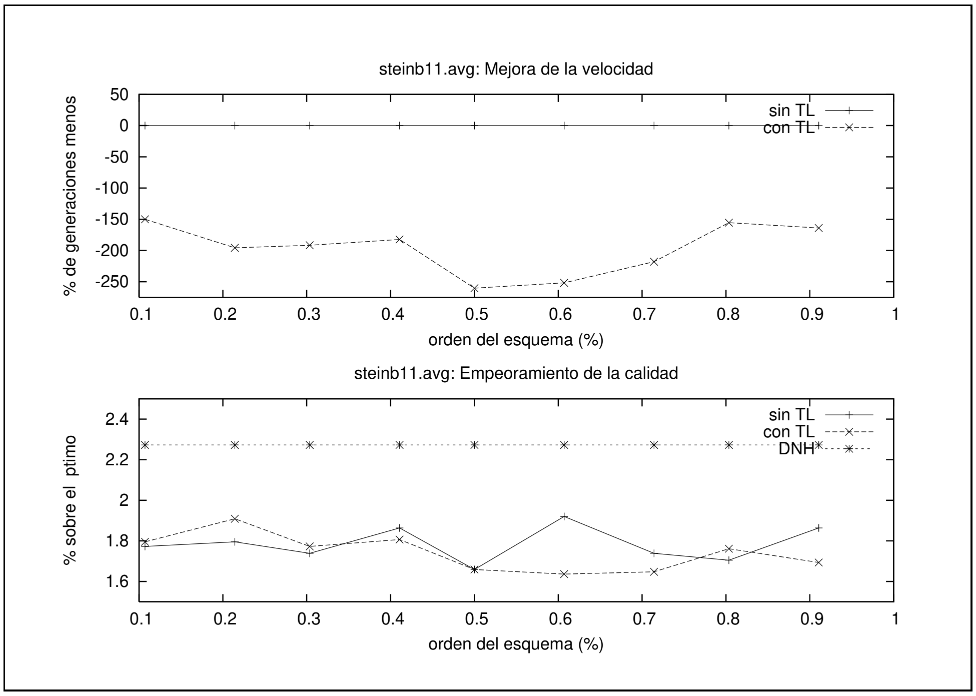


Figura 6.8: Resultados de convergencia para `steinb11.txt` con `E=0 S=1`.

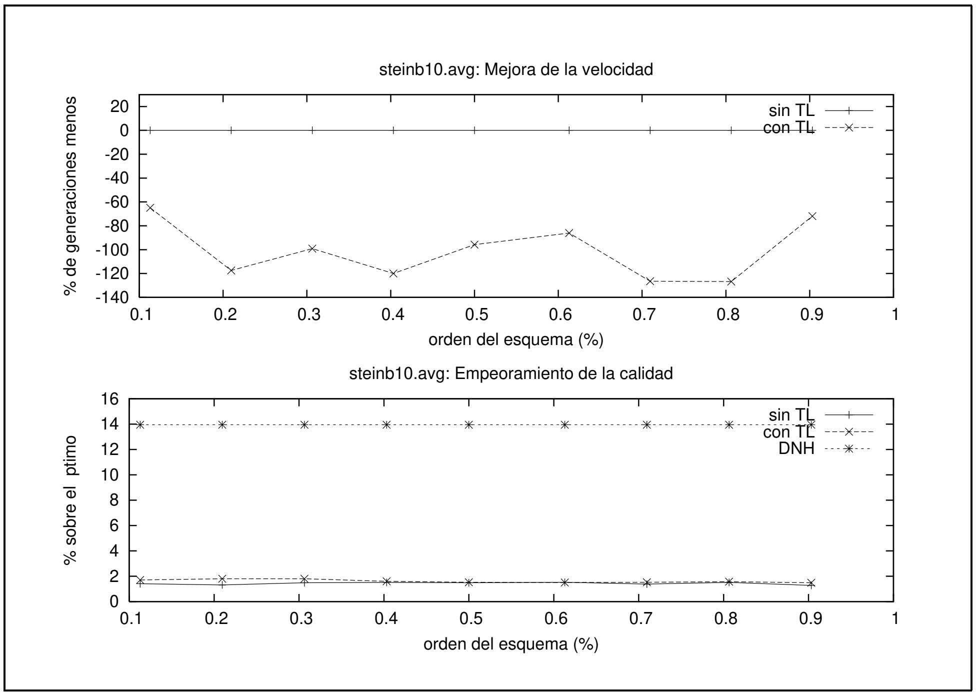


Figura 6.9: Resultados de convergencia para `steinb10.txt` con `E=1 S=1`.

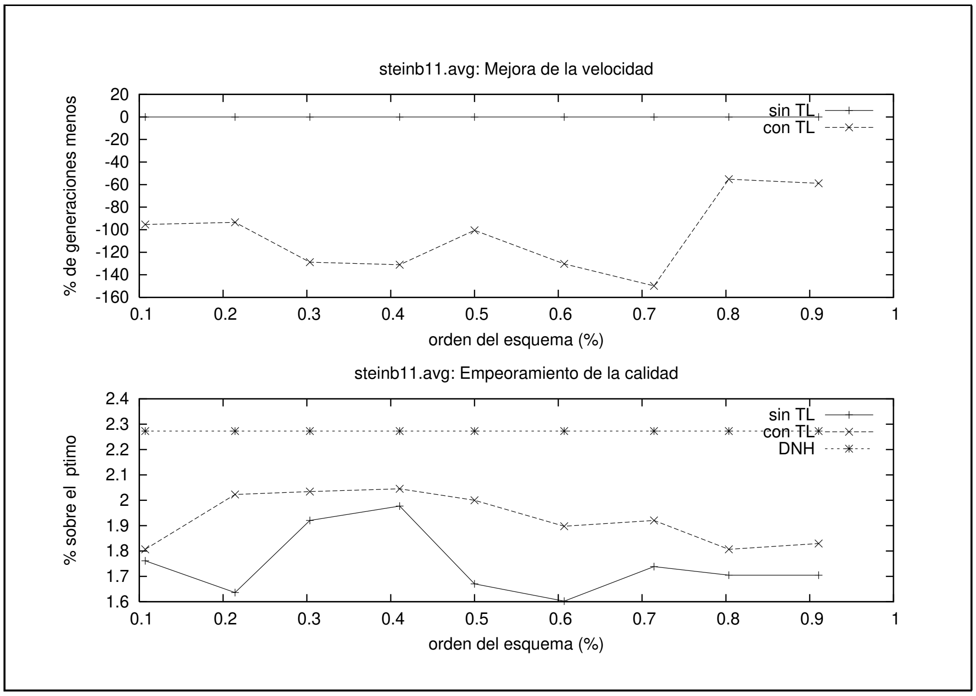


Figura 6.10: Resultados de convergencia para `steinb11.txt` con `E=1 S=1`.

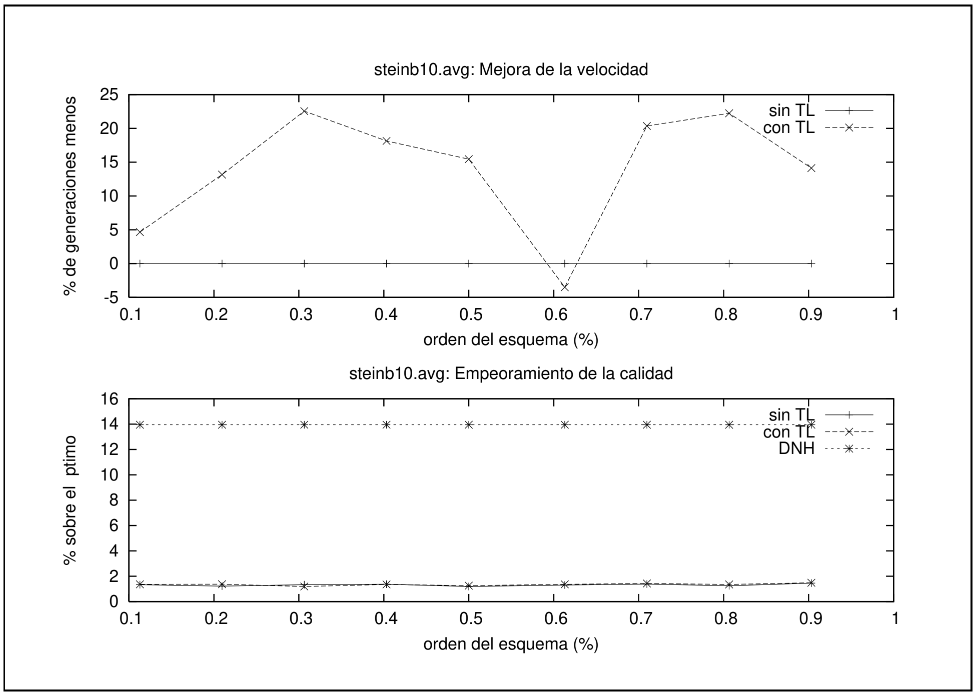


Figura 6.11: Resultados de convergencia para `steinb10.txt` con `E=1 S=0`.

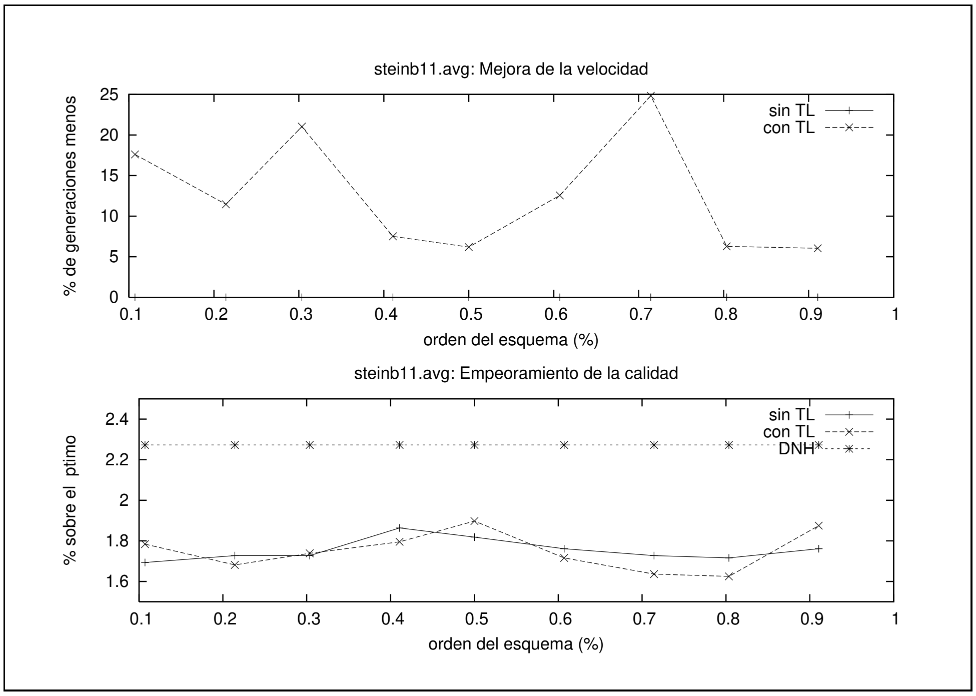


Figura 6.12: Resultados de convergencia para `steinb11.txt` con `E=1 S=0`.

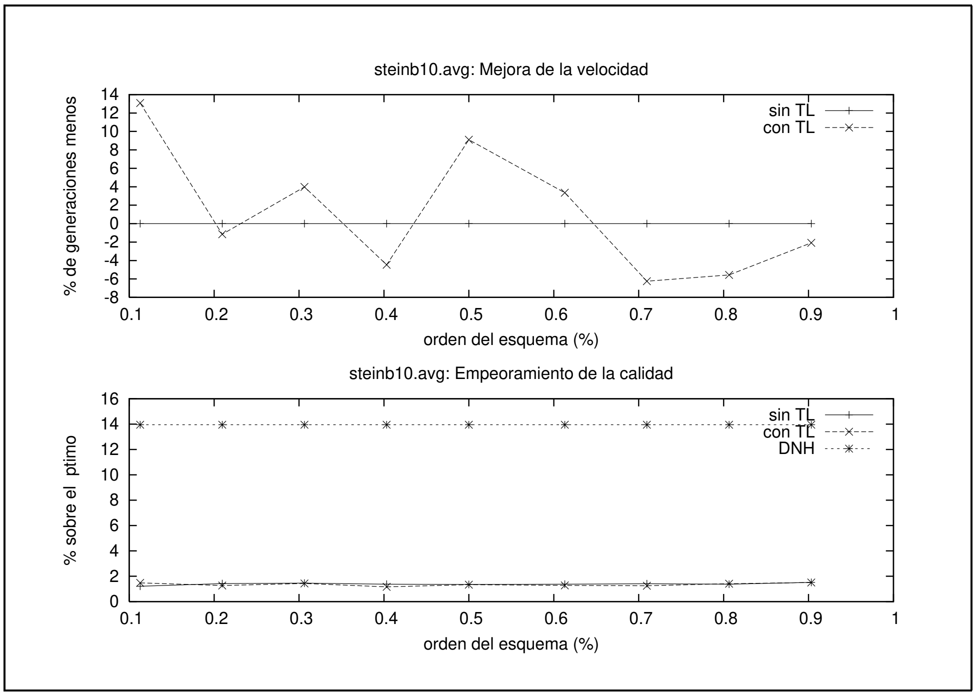


Figura 6.13: Resultados de convergencia para `steinb10.txt` con `E=0 S=0`.

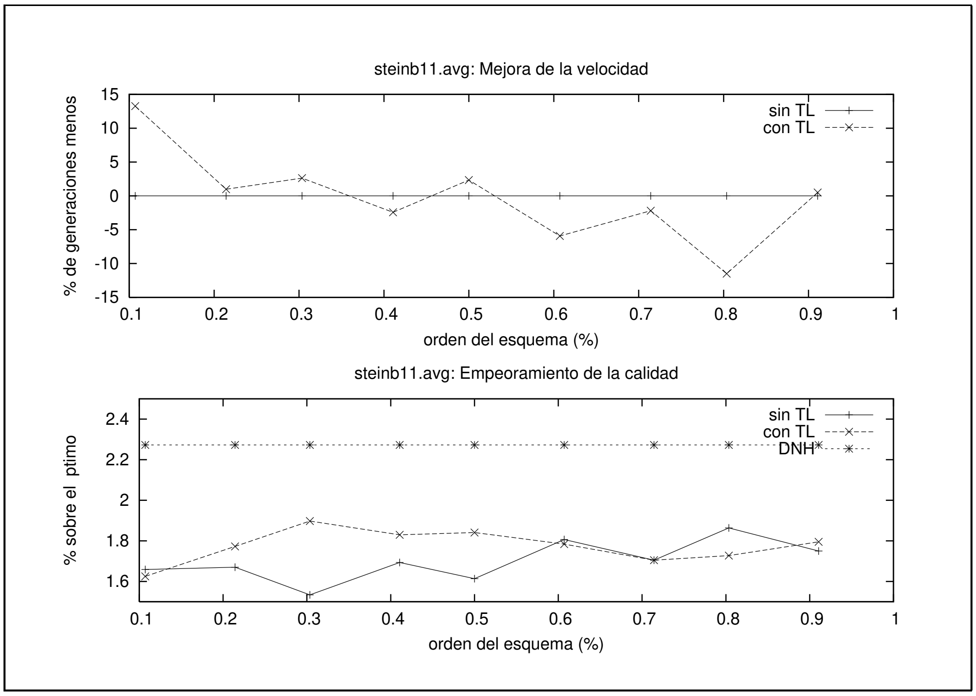


Figura 6.14: Resultados de convergencia para `steinb11.txt` con `E=0 S=0`.

El método que empleamos, con el ordenamiento sin prioridad, y sin elitismo, produce en cambio una mejora de hasta aproximadamente un 14 % en la velocidad de convergencia, sin deteriorar la calidad de las soluciones obtenidas sin la transformación lineal, y que se mantienen dentro de un rango de un empeoramiento máximo del 2 % respecto al óptimo.

La adición de elitismo lleva la mejora hasta casi un 25 % en la velocidad de convergencia, y además hace esta mejora más robusta frente al orden del esquema de referencia a partir del cual se genera la transformación lineal.

En ninguno de los casos se ha apreciado una diferencia significativa entre problemas, por lo que podemos pensar que nuestro método es válido tanto si el heurístico da una buena solución por sí solo, como si no.

6.4. Extensión del estudio a otros problemas

A continuación vamos a comprobar sobre otros problemas del mismo grupo, si los resultados presentados en la sección anterior se mantienen. Solo analizaremos problemas con el ordenamiento de las estadísticas sin dar prioridad a los "1", ya que en la primera fase del análisis la otra opción presentaba poco interés.

6.4.1. Caso `E=1 S=0`

En el problema `steinb04`, Fig. 6.16, de nuevo aparecen los dos picos de mejora de los resultados, aproximadamente en los mismos órdenes, y con una magnitud similar. También en el problema `steinb06`, Fig. 6.17, hay zonas de mejora, aunque de magnitud muy pequeña.

El resto de problemas analizados muestran empeoramiento de la velocidad en todos los casos:

- Entre un 5 % y un 25 % para el problema `steinb02`, Fig. 6.15.
- Entre un 10 % y un 50 % para el problema `steinb07`, Fig. 6.18.
- Hasta un 30 % para el problema `steinb09`, Fig. 6.19.
- Hasta un 120 % para el problema `steinb14`, Fig. 6.20.
- Entre un 5 % y un 25 % para el problema `steinb15`, Fig. 6.21.

Estos resultados no muestran correlación con el grado de dificultad que tiene el problema para el HRD, como se puede comprobar comparando los resultados con los datos del Cuadro 6.2.

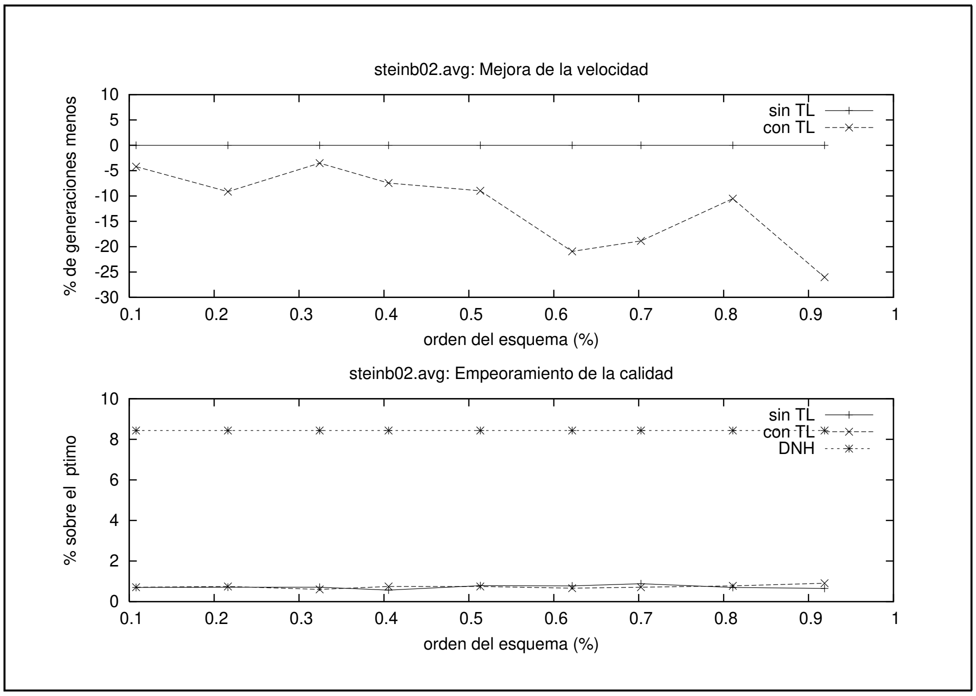


Figura 6.15: Resultados de convergencia para `steinb02.txt` con `E=1 S=0`.

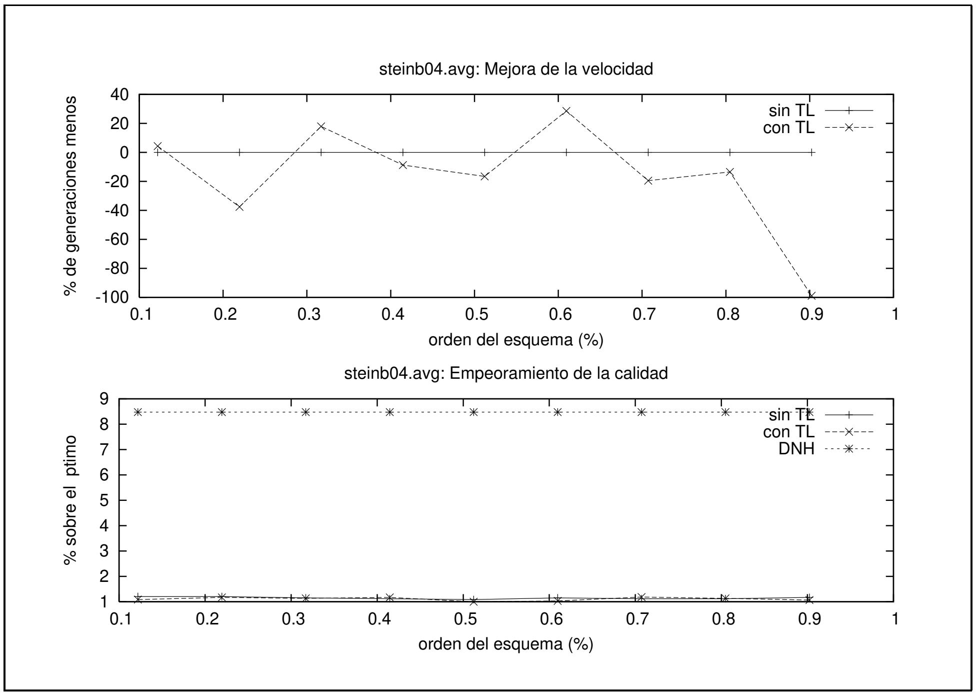


Figura 6.16: Resultados de convergencia para `steinb04.txt` con `E=1 S=0`.

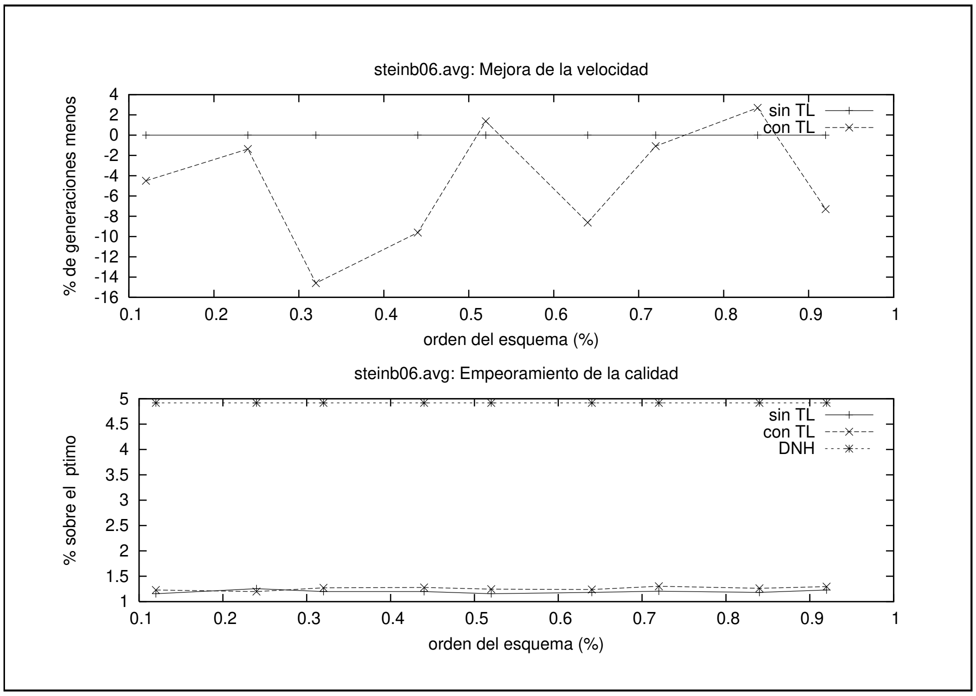


Figura 6.17: Resultados de convergencia para `steinb06.txt` con `E=1 S=0`.

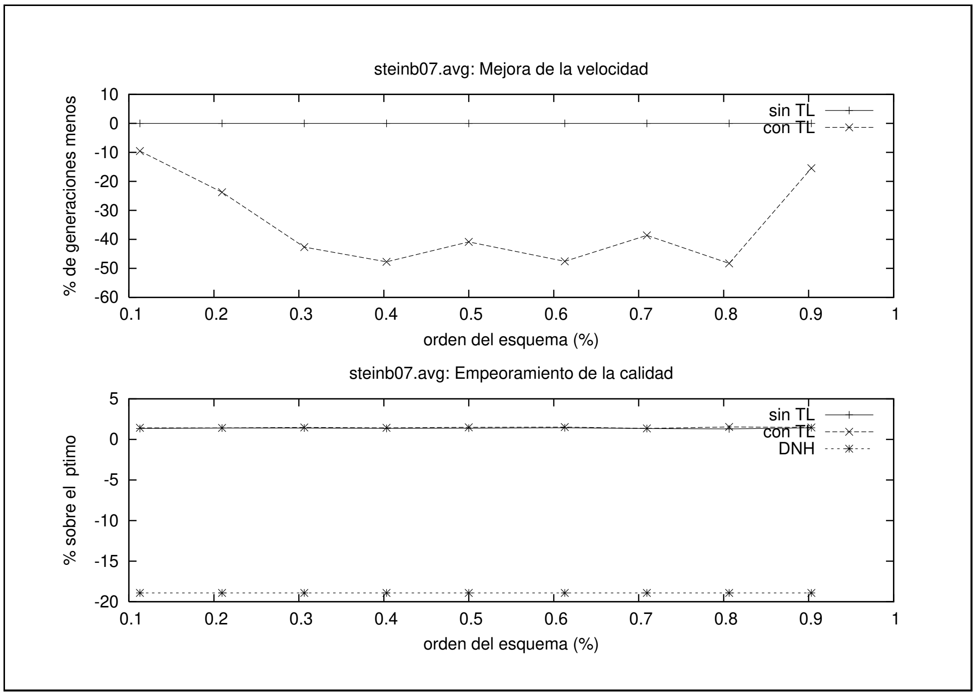


Figura 6.18: Resultados de convergencia para `steinb07.txt` con `E=1 S=0`.

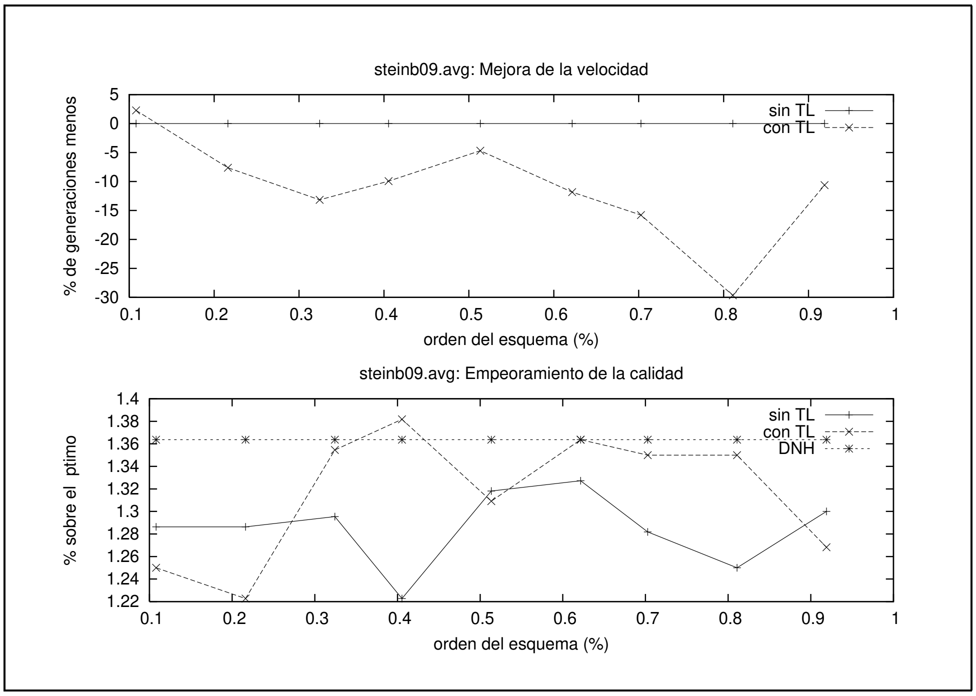


Figura 6.19: Resultados de convergencia para `steinb09.txt` con `E=1 S=0`.

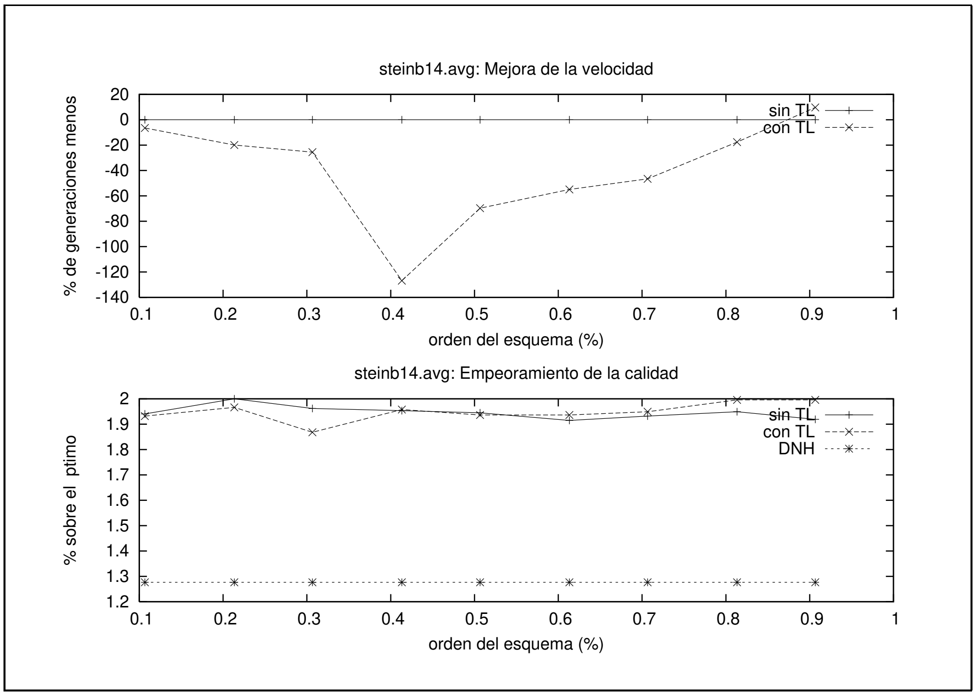


Figura 6.20: Resultados de convergencia para `steinb14.txt` con `E=1 S=0`.

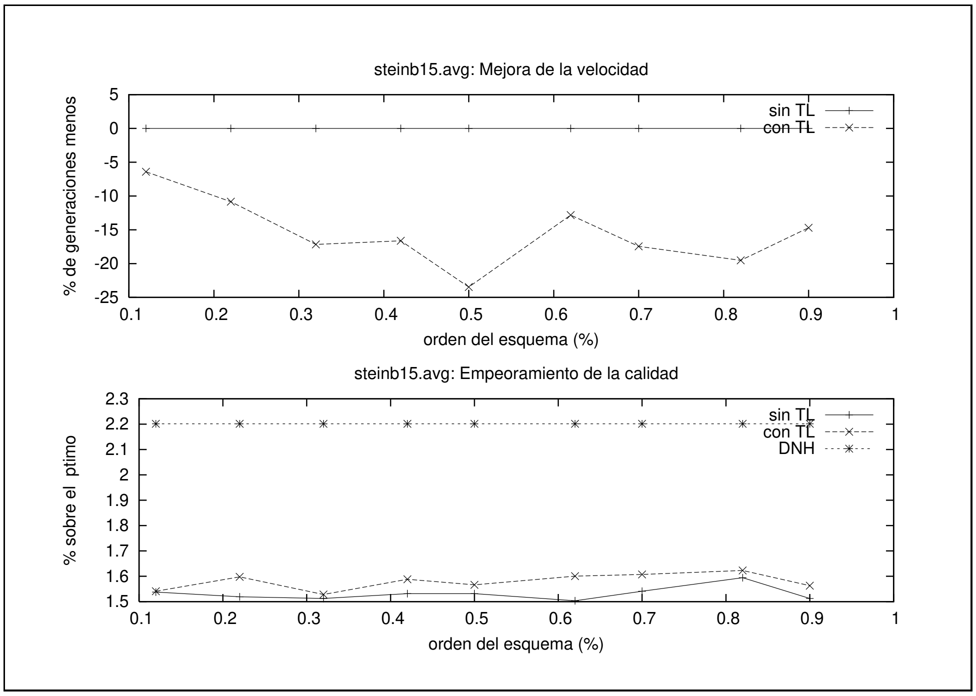


Figura 6.21: Resultados de convergencia para `steinb15.txt` con `E=1 S=0`.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Se ha implementado un algoritmo genético híbrido modificado por una transformación lineal unitaria obtenida a partir del análisis de las estadísticas de los genotipos en una población, y se ha estudiado el efecto de ésta sobre las prestaciones del algoritmo de manera que los resultados fueran estadísticamente relevantes.

El trabajo se ha desarrollado en un entorno libre, en un lenguaje muy difundido entre la comunidad científica (C++), y aprovechando bibliotecas especializadas en grafos (BGL) y algoritmos evolutivos (EO), lo que facilitará su difusión y mejora.

Los dos objetivos principales del Proyecto se han cumplido, por tanto, así como los objetivos secundarios que han surgido durante el tiempo de vida del Proyecto.

7.1.1. Sobre la Transformación lineal

La TLU diseñada ha demostrado, para el ordenamiento de estadísticas por grado de determinismo, y con reemplazamiento elitista, que es capaz de incrementar la velocidad del algoritmo genético híbrido hasta un 25 % para determinados problemas de la biblioteca OR, si se genera a partir de un esquema con un 30 % o un 70 % de sus bits fijos.

Sin embargo, para otros problemas no sólo no hay mejora en la velocidad, sino un empeoramiento importante. No ha sido posible establecer una relación entre algún parámetro del problema (particularmente la calidad de la solución hallada por el HRD) y esta mejora o empeoramiento de las prestaciones.

Cabe destacar que el diseño de la matriz permite que ésta sea invertible en todos los casos, y que su inversa sea igual a ella misma, por lo que no es necesario implementar una función específica para la inversión de matrices en módulo 2.

7.1.2. Sobre el desarrollo de un entorno de trabajo

La implementación de los algoritmos del Proyecto se realizó en un único lenguaje, C++, de amplia difusión, lo que supuso un ahorro de tiempo en la curva de aprendizaje de las personas encargadas del Proyecto. El lenguaje C++ es orientado a objetos, permite la programación genérica y es compilable, lo que lo dota de una gran versatilidad y potencia.

Se aprovecharon las bibliotecas libres `BGL` para la implementación de los tipos de datos y algoritmos sobre grafos, y `EO` para la parte del algoritmo genético. Esto nos proporcionó una base sólida para el software que hubo que crear a continuación, además de la posibilidad de sustituir a consultores especializados por la comunidad de desarrolladores.

Como plataforma se utilizó un sistema operativo GNU/Linux bajo distribución Debian, que demostró ser robusto, eficiente, con una gran variedad de programas científicos y de edición de calidad muy alta, y plenamente funcional para el trabajo en red. Esta plataforma tiene el valor añadido de ser libre y gratuita.

7.2. Trabajo futuro

Como se adelantó en la introducción a este Proyecto, se intuye que en el espectro de la función de evaluación puede encontrarse información muy importante sobre las prestaciones del algoritmo genético, pero aún no se sabe cómo obtenerla, ni cómo emplearla para mejorar las prestaciones del algoritmo genético.

Las transformaciones lineales unitarias tienen una capacidad de actuación sobre el espectro moderada, por lo que es nesario desarrollar la teoría de transformaciones no lineales.

Aunque es cierto que los algoritmos con codificación *ad hoc* están siendo investigados intensivamente, se está empezando a trabajar en serio sobre la teoría de la representación de algoritmos genéticos y a identificar los problemas con los que se encuentran.

Por lo tanto, consideramos muy interesante la línea de investigación que:

1. Determine definitivamente los supuestos bajo los que es válido el análisis del espectro de una función de evaluación, y su relación con los

distintos problemas de representación que afectan a los algoritmos genéticos.

2. Encuentre métodos rápidos y precisos para estimar el espectro.

3. Desarrolle una teoría que permita generar transformaciones no lineales para buscar buenas representaciones del genotipo.

Parte III

Anexos

Apéndice A

Presupuesto

Seguidamente se detallan los costes asociados a la realización de este Proyecto, tanto en concepto de mano de obra, como material e instalaciones. Todos los costes están dados en horas para un sólo trabajador.

El caso de las consultas con especialistas no refleja unos pagos reales, al haberse realizado en un entorno de investigación universitaria, y dada la buena voluntad de aquellos. Pero se han de diferenciar dos situaciones:

- Consultas desde el punto de vista del intercambio de conocimientos entre investigadores. En este supuesto, sí se ha asignado un coste real, ya que es equivalente al salario asignado a los ingenieros que han desarrollado el Proyecto.
- Consultas desde el punto de vista del desarrollo de software libre. En este supuesto no se ha asignado ningún coste, ya que esto refleja la situación real en el mundo laboral, como ventaja de los sistemas libres frente a los propietarios.

El coste de la hora de trabajo por persona se ha fijado en:

- Ingenieros: 25 €/persona/h.
- Consultores: 75 €/persona/h.

A.1. Documentación

A.1.1. AGH (Algoritmos genéticos híbridos)

La base teórica principal del Proyecto se encuentra en el campo de los algoritmos genéticos híbridos. Si bien la teoría básica de algoritmos genéticos se puede adquirir en cursos universitarios, el estudio de representación y espectro es puntero y requiere especialización.

1. Especialización en algoritmos híbridos (*20 h*).
2. Estudio del análisis de la representación y de espectros (*60 h*).
3. Consultas con un especialista, Dr. Franz Rothlauf, del IlliGAL y la Universität Mannheim (*8 h*, coste de consulta a especialista).

Duración: *88 h*
Coste: *2.100 €*

A.1.2. HRD (Heurístico de Red de Distancia)

Aunque el HRD es un componente del algoritmo genético híbrido, lleva asociadas una serie de tareas con suficiente entidad como para desglosarlas.

1. Especialización en teoría de grafos orientada al problema de Steiner en un grafo (*40 h*).

Duración: *40 h*
Coste: *1.000 €*

A.1.3. Biblioteca BGL (Boost Graph Library)

Dado que el HRD está compuesto por diversos algoritmos aplicados a un grafo, fue importante tomar una buena decisión inicial sobre la implementación, con vistas a la dificultad en el desarrollo, aprovechamiento de memoria, y velocidad, ya que una mala implementación del software de grafos puede llevar rápidamente a errores de ejecución, saturación de recursos, etc.

1. Decisión del lenguaje de programación –C o C++– (*16 h*).
2. Búsqueda de bibliotecas ya desarrolladas, en especial de software libre (*24 h*).
3. Decisión entre las bibliotecas favorable a BGL (*16 h*).
4. Estudio de las bibliotecas Boost y Boost Graph (*40 h*).

Duración: *96 h*
Coste: *2.400 €*

A.1.4. Biblioteca EO (Evolutionary Objects)

Al tratarse de un Proyecto con una fuerte componente de investigación, se decidió dar preferencia a la flexibilidad, estandarización y facilidad de desarrollo frente a la optimización máxima del código ejecutable. Se previeron cambios en los parámetros y operadores del AG.

1. Decisión del lenguaje de programación –C o C++– (*8 h*).
2. Búsqueda de bibliotecas ya desarrolladas, en especial de software libre (*24 h*).
3. Decisión entre la biblioteca pgapack –C– o EO –C++– (*32 h*).
4. Estudio de la biblioteca. Consultas con el especialista Marc Schoenauer, director del Equipe Evolution Artificielle et Apprentissage del INRIA y desarrollador en activo de la biblioteca (*20 h*, sin coste asociado).

Duración: *84 h*
Coste: *1.600 €*

A.1.5. Herramientas de representación gráfica

Se buscaron herramientas para la representación automática de grafos a partir de los ficheros de la biblioteca OR, y de visualización de los resultados de las simulaciones.

1. Estudio de dotty, un editor de grafos (*16 h*).
2. Estudio de gnuplot, como editor de gráficas (*16 h*).

Duración: *32 h*
Coste: *800 €*

A.1.6. Costes no incluídos

Se supone un entrenamiento previo en

- C y C++, incluida la STL (Standard Template Library).
- Teoría de grafos básica.
- Teoría de algoritmos genéticos básica.

por lo que no se computan a efectos del Presupuesto las horas necesarias para el aprendizaje de estas disciplinas.

A.2. Software

A.2.1. Diseño

1. Requisitos funcionales (*80 h*).

2. Interfaz de usuario (*60 h*).

Duración: *140 h*
Coste: *3.500 €*

A.2.2. Banco de pruebas. Biblioteca OR (Operations Research).

Para poder evaluar las prestaciones del algoritmo, es necesario contar con unos problemas estándar. Se decidió usar los de la biblioteca OR, al igual que en Esbensen [14].

- Búsqueda de un banco de pruebas contrastable –biblioteca OR– (*40 h*).

- Desarrollo de las funciones de entrada / salida según el formato de los ficheros de la biblioteca (*80 h*).

Duración: *120 h*
Coste: *3.000 €*

A.2.3. HRD (Heurístico de Red de Distancias)

Al tratarse de una biblioteca diferente a la del AGH, el desarrollo del algoritmo local se definió como una tarea independiente (salvo la interfaz común).

Dado que las bibliotecas Boost y Boost Graph implementan los tipos de datos y algoritmos para trabajar sobre los grafos y que se distribuyen gratuitamente bajo la licencia Artistic, los costes asociados al desarrollo del software de base y licencias son nulos.

- Definición de la interfaz común con el algoritmo genético y con la biblioteca OR (*24 h*).

- Implementación de la función HRD (*276 h*).

Duración: *300 h*
Coste: *7.500 €*

A.2.4. AGH (Algoritmo Genético Híbrido)

Partiendo de las clases básicas de la biblioteca EO, fue necesario ampliarlas para ajustarlas a los datos del problema, así como crear los sistemas de análisis de esquemas y transformación lineal.

Dado que la biblioteca EO implementa la estructura básica del AG y los operadores más utilizados, y que se distribuyen gratuitamente bajo la licencia LGPL, los costes asociados al desarrollo del software de base y licencias son nulos.

- Definición de la interfaz común con el HRD (*16 h*).
- Implementación de la función de descodificación e integración con el HRD (*120 h*).
- Implementación de la función de análisis de esquemas (*120 h*).
- Implementación de la función de transformación lineal (*120 h*).
- Colaboración con el desarrollador de la biblioteca para mejorarla (*16 h*).

Duración: *392 h*
Coste: *9.800 €*

A.2.5. Guiones de análisis de resultados

Se decidió implementar unos guiones que extrajeran de los ficheros de resultados los datos relevantes, en un formato que permitiera su análisis por un operador.

- Implementación del guión de iteración de simulaciones, con posibilidad de paralelización (*16 h*).
- Implementación del guión de extracción de datos de los ficheros de resultados (*24 h*).
- Implementación del guión de combinación de resultados (*16 h*).
- Implementación del guión de representación gráfica (*16 h*).

Duración: *72 h*
Coste: *1.800 €*

A.3. Pruebas de laboratorio

A.3.1. Preparación de las simulaciones

Se seleccionaron los problemas y parámetros que se iban a estudiar. También fue necesario hacer una estimación del tiempo que necesitarían las simulaciones, y del número de éstas necesario para obtener resultados representativos.

- Selección de los problemas por representatividad, tiempo de convergencia y de ejecución (*120 h*).
- Selección de los parámetros a modificar (*60 h*).
- Simulaciones previas para estimación de tiempos (*40 h*).
- Selección del número de simulaciones a realizar (*16 h*).

Duración: *236 h*
Coste: *5.900 €*

A.3.2. Simulaciones

La obtención de resultados representativos estadísticamente supuso los mayores tiempos de espera, en una actividad en la que los operadores humandos tienen una participación mínima.

- Creación de la red de los equipos necesarios para las simulaciones (*40 h*).
- Realización de las simulaciones en paralelo (*240 h*).

Duración: *280 h*
Coste: *7.000 €*

A.4. Documentación del Proyecto

A.4.1. Documentación del software

El software debe estar preparado para que un equipo de ingenieros con unos conocimientos similares a los de los integrantes del equipo original puedan retomar el trabajo en el punto que se dejó. Además un usuario distinto del desarrollador debe ser capaz de usar los programas.

- Comentario del código fuente de los programas y guiones (*40 h*).
- Documentación general del código y su interacción (*20 h*).
- Requisitos no funcionales y funcionales de los programas y guiones (*20 h*).

Duración: *80 h*
Coste: *2.000 €*

A.4.2. Memoria del Proyecto

Selección y escritura de entre toda la documentación recopilada de los temas principales y de los resultados de laboratorio.

- Introducción teórica del problema de Steiner (*16 h*).
- Introducción teórica de la teoría de AGH (*16 h*).
- Introducción teórica del espectro y la representación. Transformaciones lineales unitarias (*16 h*).
- Resultados de laboratorio (*16 h*).
- Presupuesto (*16 h*).

Duración: *80 h*
Coste: *2.000 €*

En el Cuadro A.1 puede verse un resumen de los costes asociados a tareas. El coste total es de 50.400 €, y la duración total, de 2.040 h. Contando con 20 días laborables de 8 h. al mes, esto supone 12,8 meses de trabajo.

A.5. Material y Equipos

En esta sección se detallan los costes no asociados a tareas, ni a salarios. Para la realización del Proyecto se empleó un laboratorio de 140 m^2 durante 1 año en la localidad de Leganés (Madrid). Se estima el precio del alquiler en unos 1.070 €/ mes. En él trabajaron 10 personas, de las cuales sólo una estuvo asignada al Proyecto, por lo que coste correspondiente al local queda en 1.284 €.

Los gastos asociados a mantenimiento, reparaciones y limpieza se estiman en otros 1.000 € al año.

Costes asociados a tareas		
Tarea	*Coste (horas)*	*Coste (€)*
Documentación AGH	88	2.100
Documentación HRD	40	1.000
Documentación biblioteca BGL	96	2.400
Documentación biblioteca EO	84	1.600
Documentación herramientas gráficas	32	800
Diseño de software	140	3.500
Banco de pruebas. Biblioteca OR	120	3.000
Implementación HRD	300	7.500
Implementación AGH	392	9.800
Implementación guiones	72	1.800
Preparación de las simulaciones	236	5.900
Simulaciones	280	7.000
Documentación del software	80	2.000
Memoria del Proyecto	80	2.000
Total	2.040	50.400

Cuadro A.1: Resumen de costes asociados a tareas.

Para las fases de diseño e implementación se empleó un único ordenador portátil PIII 866 MHz 384 Mb SDRAM con un coste de 1.954 €. Al término del Proyecto, su valor en el mercado era de 450 €, por lo que el costo asociado es de 1.504 €.

Para la fase de simulación, se aprovecharon el resto de equipos del laboratorio; cada uno de estos 9 equipos costó 1.500 € y está asignado a un proyecto. Dado el envejecimiento del hardware, cada equipo se puede utilizar sólo durante 3 años antes de tener que cambiarlo, momento en el que su valor residual es despreciable, por lo que se destinan a obra social, como colegios u ONG. Por lo tanto el coste asociado a este concepto para el Proyecto es de 1.500 / (9 + 1) / 3 € = 50 €.

Para el gasto de la red interna del laboratorio y un acceso para empresas de 2 Mbps a internet, y amortización del alta e instalación a 5 años, estimamos un coste anual de 500 € de la fracción asignable a este Proyecto.

La documentación se obtuvo principalmente de Internet, del servicio de préstamos interbibliotecario de la Universidad, de la biblioteca y de la compra de libros. Estimamos estos costes en 300 €.

El sistema operativo empleado fue GNU/Linux, y todo el software es libre, por lo que los costes por licencias son nulos.

En el Cuadro A.2 puede verse un resumen de los costes de material y equipos.

Costes de material y equipos	
Concepto	*Coste (€)*
Laboratorio	1.284
Mantenimiento, reparaciones . . .	1.000
Ordenador portátil PIII 866 MHz	1.504
Equipos de simulación	50
Intranet y acceso a Internet	500
Documentación	300
Total	4.638

Cuadro A.2: Resumen de costes de material y equipos.

A.6. Coste y duración total del Proyecto

Sumando los costes anteriores, el presupuesto total para el Proyecto es de CINCUENTA Y CINCO MIL TREINTA Y OCHO € (55.038 €).

La duración total del Proyecto fue de 12,8 meses.

Coste total del Proyecto	
Concepto	*Coste (€)*
Costes asociados a tareas	50.400
Costes de material y equipos	4.638
Total	55.038

Cuadro A.3: Coste total del Proyecto.

En las Fig. A.1 y Fig. A.2, se muestra la carga relativa de cada apartado en el coste y duración totales del Proyecto.

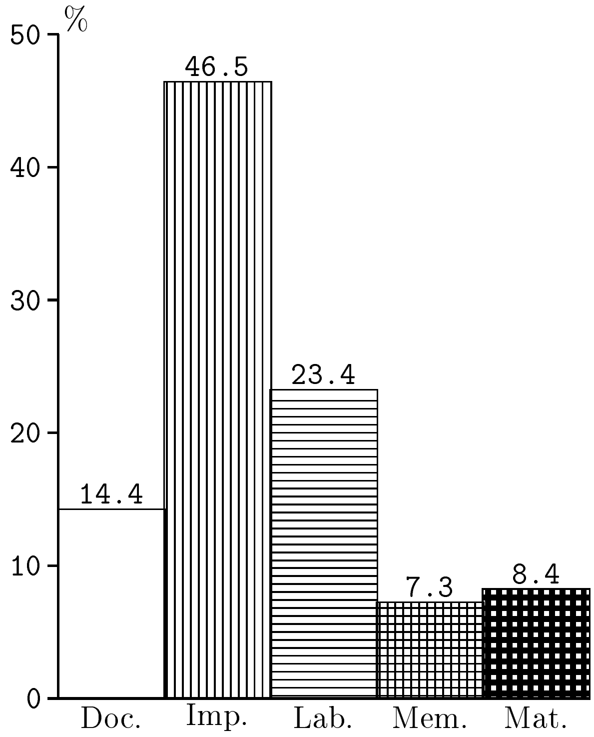


Figura A.1: Reparto de costes del Proyecto.

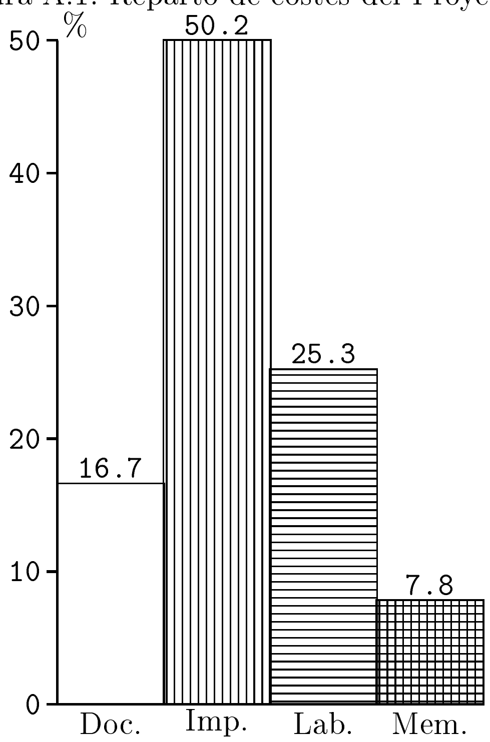


Figura A.2: Duración relativa de las tareas.

Apéndice **B**

Introducción a la teoría de la complejidad

Presentamos a continuación algunos conceptos y definiciones básicos de la teoría de la complejidad [27], que pueden resultar de gran ayuda a la hora de entender la *dificultad* de resolver el problema de Steiner. Los tipos de problemas con los que nos enfrentamos normalmente son de uno de estos tipos:

- ***Decisión***: Se requiere una respuesta afirmativa o negativa a una pregunta. Por ejemplo, si existe un árbol de Steiner para la instancia del problema considerada con un coste menor que una constante C.
- ***Optimización***: Dentro de un conjunto de estructuras de un tipo determinado, buscamos la óptima según un criterio. Por ejemplo, buscar un árbol de Steiner de coste mínimo. Un problema de optimización es al menos tan difícil de resolver como uno de decisión.
- ***Evaluación***: Se busca el valor de una solución óptima, sin necesidad de hallar también la solución de forma explícita. Por ejemplo, hallar el coste de un árbol de Steiner de coste mínimo.

Los problemas para los que existe un algoritmo de complejidad polinomial, esto es, $\mathcal{O}(f)$, para un polinomio f del tamaño de los datos de entrada, que los resuelve, se llaman *polinómicos* o *fáciles*. Los algoritmos polinómicos reciben el nombre de *buenos* o *eficientes*.

Definición B.1 (*Problemas P*) *Se llama* P *(de* polinómicos*) a la clase de todos los problemas de decisión polinómicos.*

Definición B.2 (*Problemas PN*) *Se llama* PN *(de* polinómicos no deterministas*) a la clase de todos los problemas de decisión para los que una respuesta afirmativa se puede verificar en tiempo polinomial.*

En este caso no basta con una respuesta afirmativa ("sí existe una solución que cumple la condición") o negativa ("no existe ninguna solución que cumpla la condición"), sino que necesitamos un modo de verificar la corrección de la respuesta; esto supone generalmente comprobar que la estructura de la solución es la adecuada, y que efectivamente cumple la condición impuesta.

Definición B.3 (Problemas Co-PN) *Se llama* Co-PN *a la clase de todos los problemas de decisión para los que una respuesta negativa se puede verificar en tiempo polinomial.*

La relación P $\subset$ PN $\cap$ Co-PN es obvia a partir de las definiciones anteriores, y llegamos por fin a la siguiente conjetura, y a la definición de problemas PN completos.

Conjetura B.4 *Se piensa que $P \neq PN$ (y que $PN \neq$ Co-PN).*

Definición B.5 (*Problemas PN completos*) *Se llaman* PN completos *a los problemas que son de PN y cuya resolución polinomial implicaría que todos los demás problemas de PN son también resolubles de forma polinómica.*

Los problemas (de decisión) PN completos pueden, por ejemplo, ser lo suficientemente *sencillos* como para comprobar una respuesta afirmativa en tiempo polinomial, pero no tanto como para encontrar un algoritmo polinómico que los resuelva, a menos que la conjetura B.4 no se cumpla.

Definición B.6 (*Problemas con dificultad PN*) *Se llama* problema con dificultad PN *a aquel para el que, sin ser de decisión, su resolución polinómica implicaría que $P = PN$. En particular, cualquier problema de optimización que se corresponde con un problema de decisión PN completo lo es.*

En resumen y de forma práctica, nos encontramos con problemas en P, para los que se pueden encontrar algoritmos polinómicos que los resuelven, y problemas en PN, *completos* o de *dificultad PN* (de decisión u optimización, respectivamente), para los que según la conjetura B.4 no existen tales algoritmos, por lo que se les llama *problemas difíciles* o *irresolubles.*

Esto no quiere decir, por supuesto, que no existan algoritmos para resolverlos de forma exacta, sino que éstos tienen complejidad exponencial en el tamaño de los datos de entrada, por lo que se considera que no resuelven *bien* el problema. De esta forma llegamos a la búsqueda de métodos no exactos alternativos para obtener soluciones *lo suficientemente buenas*, lo que se suele entender como una relación entre el tiempo que se necesita para obtener la solución, la calidad de ésta, y la complejidad del sistema.

Los métodos no exactos consisten siempre en un muestreo inteligente del espacio de búsqueda, o en un muestreo de un espacio transformado. Pueden consistir en la aplicación del sentido común, como en el caso del HRD, lo que se conoce como *métodos heurísticos*, basarse en los mecanismos evolutivos de la naturaleza, como en el caso de los algoritmos genéticos, etc. Su importancia viene determinada por la gran profusión de problemas del mundo real que son PN completos o de dificultad PN.

Apéndice C

Estudio previo de matrices unitarias

Definición C.1 (Matriz binaria) *Se llama matriz binaria a la matriz* $A = ((a_{ij})),\ a_{ij} \in \{0,\ 1\}$. *Como caso particular, se llama vector binario a una matriz binaria que tenga una de sus dimensiones igual a 1.*

Definición C.2 (Grado de un vector binario) *Se llama grado de un vector binario* v, *a la función* $\delta(v)$ *que determina el número de las componentes de* v *que son iguales a 1.*

Lema C.3 *Sea* A *una matriz binaria de dimensión* $n \times n$, *tal que* $A = B + \mathbb{I}$, *donde* $\mathbb{I}$ *es la matriz unidad y sea* d *un vector binario de longitud* n, *tal que* B *es una matriz binaria* $B = ((b_{ij}))$ *con* $b_{ij} = d_i$. *Entonces,*

$$\delta(d) \text{ es par} \Leftrightarrow \exists\ A^{-1} = A$$

Dem:

Sea $s \subset \mathbb{N}^{\delta(d)}$ *el vector ordenado de los índices de* d *tal que* $i \in s \Leftrightarrow d_i = 1$, *y sea* $t \subset \mathbb{N}^{n-\delta(d)}$ *el vector el vector ordenado de los índices de* d *tal que* $i \in s \Leftrightarrow d_i = 0$. *Sea* $p = [s\ t]$ *una permutación de las columnas de* A. *Entonces, para alguna matriz binaria* P,

$$C = p(A) = \begin{bmatrix} M & \mathbf{0} \\ \mathbf{1} & \mathbb{I} \end{bmatrix} = P \cdot A \cdot P^T$$

donde $\mathbf{0}$ *y* $\mathbf{1}$ *son dos matrices donde los elementos son, respectivamente, todos 0 y 1, y* M *es una matriz* $\delta(d) \times \delta(d)$ *de la forma*

$$M = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix}$$

Vamos a hacer ahora un cambio de cuerpo, del de los números $\{0,\ 1\}$ *a los números* $\mathbb{Z}$*. El determinante de esta matriz vale*

$$det_{\mathbb{Z}}(M) = (-1)^{\delta(d)+1}(\delta(d) - 1)$$

Por la proyección $b : \mathbb{Z} \mapsto \{0,\ 1\}$,

$$det(M) = b(det_{\mathbb{Z}}(M)) = \begin{cases} 0 & si\ det_{\mathbb{Z}}(M)\ es\ par \\ 1 & si\ det_{\mathbb{Z}}(M)\ es\ impar \end{cases}$$

lo que es equivalente a decir

$$det(M) = \begin{cases} 0 & si\ \delta(d)\ es\ impar \\ 1 & si\ \delta(d)\ es\ par \end{cases}$$

Por lo tanto, la matriz M *es invertible si y sólo si* $\delta(d)$ *es par. Esta condición es necesaria para el paso siguiente, que consiste en obtener la matriz inversa de* C*. Para ello notamos que dada la matriz* C *y siendo* M *una matriz invertible, la inversa de* C *es*

$$C^{-1} = \begin{bmatrix} M^{-1} & \mathbf{0} \\ \mathbf{1} \cdot M^{-1} & \mathbb{I} \end{bmatrix}$$

lo que se demuestra operando

$$\begin{aligned} C \cdot C^{-1} &= \begin{bmatrix} M & \mathbf{0} \\ \mathbf{1} & \mathbb{I} \end{bmatrix} \cdot \begin{bmatrix} M^{-1} & \mathbf{0} \\ \mathbf{1} \cdot M^{-1} & \mathbb{I} \end{bmatrix} \\ &= \begin{bmatrix} M \cdot M^{-1} & \mathbf{0} \\ \mathbf{1} \cdot M^{-1} + \mathbb{I} \cdot \mathbf{1} \cdot M^{-1} & \mathbb{I} \cdot \mathbb{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbb{I} \end{bmatrix} \\ &= \mathbb{I} \end{aligned}$$

Demostramos ahora que bajo la condición de invertibilidad $M^{-1} = M$*, para lo hacemos el producto* $M \cdot M$ *en el caso general,*

$$K = M \cdot M = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix}$$

Por inspección comprobamos que sólo hay dos tipos de productos escalares para obtener los elementos k_{ij}. Cuando $i = j$, $k_{ij} = \mod(\delta(d)-1,\ 2)$; como $\delta(d)$ es par, $k_{ii} = 1$. En cambio, cuando $i \neq j$, $k_{ij} = \mod(\delta(d)-2,\ 2)$; como $\delta(d)$ es par, $k_{ij} = 0$. Por lo tanto,

$$K = M \cdot M = \mathbb{I}$$

y $M = M^{-1}$.

Sustituyendo este resultado en la expresión de C^{-1} obtenemos que

$$C^{-1} = \begin{bmatrix} M & \mathbf{0} \\ \mathbf{1} \cdot M & \mathbb{I} \end{bmatrix}$$

Si llamamos D al bloque inferior izquierdo de C^{-1},

$$\begin{aligned} D &= \mathbf{1} \cdot M \\ &= \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \dots & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix} \end{aligned}$$

podemos comprobar cómo el producto en el caso general es $d_{ij} = \delta(m_j)$, donde m_j son las columnas de M. Dado que cada columna contiene un único cero y que las dimensiones de M son $\delta(d) \times \delta(d)$, siendo $\delta(d)$ un número par, entonces $d_{ij} = 1$ para todos $i,\ j$,

$$D = \mathbf{1}$$

Con una nueva sustitución en la expresión de C^{-1} llegamos a la igualdad

$$C^{-1} = \begin{bmatrix} M & \mathbf{0} \\ \mathbf{1} & \mathbb{I} \end{bmatrix} \Rightarrow C^{-1} = C$$

Recordando ahora que

$$C = P \cdot A \cdot P^T$$

buscamos la inversa, y dado que $P^{-1} = P^T$,

$$\begin{aligned} C^{-1} &= (P \cdot A \cdot P^T)^{-1} \\ &= (P^T)^{-1} \cdot A^{-1} \cdot P^{-1} \\ &= P \cdot A^{-1} \cdot P^T \end{aligned}$$

y que $C = C^{-1}$ *obtenemos que* $A = A^{-1} \Leftrightarrow \delta(d)$ *es par, q.e.d.*
$\square$

Ilustramos este lema con un ejemplo para el caso $n = 6$ y $d = [0\ 1\ 1\ 0\ 1\ 1]$,

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Apéndice D

Glosario de términos español castellano – inglés

actas: proceedings

adaptación compartida: fitness sharing

algoritmo escalador: hill-climber algorithm

algoritmo genético: genetic algorithm

analizador sintáctico: parser

árbol de Steiner mínimo: minimum Steiner tree

árbol generador: spanning tree

atestamiento: crowding

biblioteca de funciones: function library

distancia de definición: defining length

etiqueta (de un gen del genotipo): tag

fenotipo: phenotype

función de adaptación: fitness function. Entre las muchas traducciones disponibles para *fitness*, como aptitud, calidad, ¡estado de forma!, ... *adaptación* nos parece la mejor, no sólo porque describe correctamente el objeto al que se aplica (sustantivo de *fit*, como en *the survival of the fittest*), sino porque es un término muy usado en el ámbito de las teorías de la evolución. Es cierto que el uso de *función de fitness* está muy extendido, pero en este caso la traducción al español castellano es natural y simple, por lo que no vemos la necesidad.

función engañosa: deceptive function

función objetivo: objective function

gazapo en un programa: program bug

genotipo: genotype

grafo hueco: sparse graph

guión: script

heurístico de red de distancias: distance network heuristic (DNH)

hipótesis de los bloques de construcción: building block hypothesis

línea de instrucciones u órdenes: command line

mecanismo de escalada: hill-climbing mechanism

Muestreo Universal Estocástico: Stochastic Universal Sampling

métodos de nichos: niching methods

operador de cruce: crossover operator

operador de inversión: inversion operator

operador de mutación: mutation operator

operador de selección: selection operator

partición de hiperplano: hyperplane partition

paseo por un grafo: walk in a graph

pila de Fibonacci: Fibonacci heap

plantilla de similitud: similarity template

población de los hijos: children population

población de los padres: parents population

principio de los bloques de construcción significativos: principle of meaningful building blocks

población de vástagos: offsprings population

problema del árbol generador mínimo: minimum spanning tree problem

problema del camino más corto: shortest path problem

problema difícil: hard problem

problema irresoluble: intractable problem

problemas con dificultad PN: NP-hard problems

problema de la red de Steiner: Steiner Network Problem (SNP)

problema de Steiner en un grafo: Steiner Problem in a Graph (SPG)

problemas P (polinómicos): P (polynomial) problems

problemas PN (polinómicos no deterministas): NP (non-deterministic polynomial) problems

problemas PN completos: NP-complete problems

rastro en un grafo: trail in a graph

retroalimentación: feedback

selección por ruleta: roulette wheel selection

selección por clasificación: ranking selection

selección por torneo determinista: deterministic tournament selection

selección por torneo estocástico: stochastic tournament selection

sesgo: bias

sii (si y sólo si): iff (if and only if)

tasa de prestaciones (TP): perfomance ratio (PR)

toma y daca: trade off

Bibliografía

[1] D. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, 1987.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Reading, Massachusetts, 1983.

[3] R. K. Ahuja, K. Mehlhorn, J.B. Orlin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37:213–223, 1990.

[4] J. D. Bagley. *The behaviour of adaptive systems which employ genetic and correlation algorithms.* Doctoral dissertation, Univ. of Michigan.

[5] J. E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[6] O. Boruvka. O jistém problému minimálním. *Acta Societ. Scient. Natur. Moravicae*, 3:37–58, 1926.

[7] O. Boruvka. Príspevek k resení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnicky obzor*, 15:153–154, 1926.

[8] T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[9] Y. Davidor. Epistasis variance: A viewpoint on GA-hardness. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 23–25. Morgan Kaufmann, 1991.

[10] K. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Ann Arbor, MI, EE.UU., 1975.

[11] D. Z. Du and F. Hwang. An approach for proving lower bounds: Solution of Gilbert-Pollak's conjecture on Steiner ratio. In *Proc. 31st Annual Symp. on Foundations of Computer Science*, pages 76–85, Los Alamitos, California, IEEE Computer Society, 1990.

[12] D. Z. Du and F. Hwang. The Steiner ratio. In *Proc. National Acad. of Sciences*, volume 87, pages 9464–9466, EE.UU., 1990.

[13] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu. *Evolutionary Computation.* International Series on Computational Intelligence. CRC Press, Boca Raton, 2000.

[14] Henrik Esbensen. Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm. *NETWORKS: Networks: An International Journal*, 26:1069–1072, 1995.

[15] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses on improved network optimization algorithms. *Journal of the ACM*, 48:533–551, 1987.

[16] M. R. Garey and D. S. Johnson. The Rectilinear Steiner Problem is NP-Complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

[17] E. N. Gilbert and H. O. Pollak. Steiner Minimal Trees. *SIAM Journal of Applied Mathematics*, 16:1–29, 1968.

[18] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[19] David E. Goldberg. Construction of high-order deceptive functions from low-order walsh coefficients. 5:35–48, 1992.

[20] David E. Goldberg. *The Design of Innovation*, volume 7 of *Genetic Algorithms and Evolutionary Computation.* 2002.

[21] J. J. Grefenstette. Deception considered harmful. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann, San Mateo, CA, 1993.

[22] J. J. Grefenstette and J. Baker. How genetic algorithms work: A critical look at implicit parallelism. In *Proc. 3rd International Conference on Genetic Algorithms.* Morgan-Kaufmann, 1989.

[23] J. J. Greffenstette, editor. *Proc. of an International Conference on Genetic Algorithms and Their Applications*, Hillsdale, NJ, EE.UU., 1985. Lawrence Erlbaum Associates.

[24] J. J. Greffenstette. Representation dependencies in genetic algorithms. 1985.

[25] J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, EE.UU., 1975.

[26] F. K. Hwang, D. S. Richards, and P. Winter. The Steiner Tree Problem. 1992.

[27] Dieter Jungnickel. *Graphs, Networks and Algorithms.* Algorithms and Computation in Mathematics. Springer, Berlín, second edition, 2002.

[28] A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith. Solving the Graphical Steiner Tree Problem Using Genetic Algorithms. *Journal of the Operational Research Society*, 44(4):397–406, 1993.

[29] H. Kargupta. A striking property of genetic code-like transformations. Technichal Report EECS-99-04, EECS School, Washington State University, 1999.

[30] H. Kargupta and B. H. Park. Gene expression and fast construction of distributed evolutionary representation. *Evol. Computing*, 9(1):45–68, 2000.

[31] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Plenum Press, Nueva York, 1972.

[32] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 2(030), 1995.

[33] T. Koch, A. Martin, and S. Voss. SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000. available from SteinLib `http://elib.zib.de/steinlib`.

[34] J. B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. In *Proc. AMS*, volume 7, pages 48–50, 1956.

[35] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, Nueva York, 1976.

[36] G. E. Liepins and M. D. Vose. Representational issues in genetic optimization. *Journal of Experimental & Theoretical Artificial Intelligence*, 2:101–115, 1990.

[37] Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villacañas, and Hans-Paul Schwefel, editors. *Parallel Problem Solving from Nature - PPSN VII*, number 7, Granada, España, September 2002. Springer.

[38] C. C. Palmer. *An approach to a problem in network design using genetic algorithms.* Unpublished PhD thesis, Polytechnich University, Troy, NY, EE.UU., 1994.

[39] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Techn. J.*

[40] Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *Symposium on Discrete Algorithms*, pages 770–779, 2000.

[41] R. S. Rosenberg. *Simulation of genetic populations with biochemical properties.* Doctoral dissertation, Univ. of Michigan.

[42] Isabel Rosseti, Marcus Poggi de Aragão, Celso C. Ribeiro, Eduardo Uchoa, and Renato F. Werneck. New benchmark instances for the steiner problem in graphs.

[43] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* 2002.

[44] Günter Rudolph. Convergence of evolutionary algorithms in general search spaces. In *Proc. of 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, pages 50–54, Nayoya University, Japan, May 1996.

[45] Sancho Salcedo Sanz. *Técnicas Espectrales en la Mejora de Algoritmos Genéticos.* PhD dissertation, Univ. Carlos III de Madrid, Departamento de Teoría de la Señal y Comunicaciones, 2002.

[46] J. D. Schaffer. Some Effects of Selection Procedures on Hyperplane Sampling by Genetic Algorithms. In *Genetic Algorithms and Simmulated Annealing.* L. Davis, ed. Pitman, 1987.

[47] Marc Schönauer. *EO Tutorial*, v0.97 edition, May 2002.

[48] J. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library (BGL)*, 2000.

[49] G. Syswerda. Uniform Crossover in Genetic Algorithms. In *Proc. 3rd International Conf on Genetic Algorithms*, pages 2–9. Morgan-Kaufmann, 1989.

[50] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Japonica*, 24:573–577, 1980.

[51] The PARI Group. *PARI/GP, Version 2.1.4*. Bordeaux, 2000. available from `http:://www.parigp-home.de/`.

[52] C. Thornton. The building block fallacy, 1997. available from `http:://www.csu.edu.au/ci/vol04/thornton/building.htm/`.

[53] Darrell Whitley. The Genitor Alogorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best. In *Proc. of the 3rd International Conference on Genetic Algorithms*, pages 116–121, 1989.

[54] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[55] Alexander Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. Technical Report CS-96-06, 1996.