Máster en Sistemas Telemáticos e Informáticos
Curso 2007-2008

Proyecto de fin de máster

Metaheurísticas aplicadas a problemas de ordenación lineal sobre grafos

Autor: Miguel Ángel Domínguez Coloma
Tutores: Juan José Pantrigo, Abraham Duarte

Agradecimientos

Quería agradecer el ánimo y el apoyo que me han dado mis dos tutores Juan José Pantrigo y Abraham Duarte. No solo por la posibilidad de poder realizar este proyecto junto con ellos, sino por la simpatía que me han ofrecido y los consejos que me han dado. Otro agradecimiento a los miembros de GAVAB, que me parece un grupo genial en donde se apoyan unos a otros y siempre hay buen humor.

Gracias a todos los amigos que he conocido a lo largo de este año en mis viajes por Europa, por compartir los idiomas, culturas e historias. Siempre ofreciendo ratos buenos y momentos inolvidables. Por ellos he podido deshacerme del estrés generado durante todo el año con los proyectos. A Sekhyun Shim, Yeongeun Kwon, Caroline Dubois, Caterina Carosi, Laia Carbonera y a Laure Jaehrling que me ha apoyado durante estos últimos días. A Juan Castro, mi compañero de aventuras por Irlanda. Otro agradecimiento a todas aquellas personas que he conocido expontáneamente y no recuerdo sus nombres.

A mi familia por su apoyo constante, de nuevo, en este máster. Y una especial dedicatoria a mi perro Rufo, que en el último trabajo no le añadí.

Resumen

Los problemas de etiquetado lineal de grafos son un conjunto de problemas en los cuales la meta es conseguir etiquetar cada vértice de un grafo con un número real. Existen varios problemas que siguen este modelo y tienen objetivos diferentes. Por ejemplo, intentar que la suma de las diferencias de etiquetas de los nodos sea la mínima, como es en el caso del problema del *Etiquetado Lineal Minimo*. Estos problemas, tratados desde hace décadas siguen teniendo un campo abierto de posibilidades e investigación, en la que se siguen obteniendo técnicas para reducir la complejidad de ejecución y tiempo para darles solución.

La mayoría de problemas de etiquetado de grafos son problemas NP-Duros o NP-Completos, los cuales nos imposibilitan la obtención de un algoritmo exacto que nos proporcione la solución óptima en poco tiempo. No obstante, durante la historia de estos problemas, se han aplicado técnicas heurísticas y metaheurísticas que nos proporcionan soluciones factibles y de gran calidad.

Durante el transcurso de esta memoria, el lector podrá contemplar el panorama histórico y actual de las técnicas que han sido propuestas para la resolución de los problemas de etiquetado en grafos. Además podrá hacer un seguimiento de dos técnicas metaheurísticas que aplicaremos a dos tipos de problemas, Recocido Simulado o *Simulated Annealing* y GRASP (*Greedy Randomized Adaptive Search Procedure*, que aplicaremos para obtener resultados óptimos en muy poco tiempo. También podrá contemplar los pseudo-códigos utilizados en la implementación para su fácil compresión o implementación futura.

Los resultados obtenidos mediante la aplicación de las metaheurísticas nos ofrecerán resultados muy cercanos al estado del arte actual. Haremos una comparación basándonos en la parametrización de las metaheurísticas, creando varios constructivos, métodos de mejora y otros mecanismos que necesitemos. De esta forma, indicaremos cuales son los mejores resultados obtenidos, comparándolos con los que no nos ofrecen tan buenas soluciones y posteriormente debatir el por qué de estas soluciones.

En conclusión, esta memoria trata de facilitar la comprensión tanto de la descripción, historia y estado del arte de los problemas de etiquetado lineal de grafos como la teoría e implementación de varias metaheurísticas aplicadas a problemas reales.

Descriptores

diseño lineal de grafos, metaheurísticas, grasp, simulated annealing, constructivos, métodos de mejora, análisis numérico, circuitos VLSI, dibujo de grafos, procesamiento paralelo y distribuido, algoritmia, problemas np-completos, aleatorización, teoría de grafos.

Índice general

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

1.1 Descripción del problema

Los problemas de distribución de grafos son una clase particular de problemas de optimización combinatoria de los cuales la meta es encontrar un etiquetado linear dado un grafo inicial y buscar una función optimizada para satisfacer la resolución al problema. Un etiquetado lineal es un grafo en el cual sus nodos están etiquetados mediante distintos números enteros. Existe un gran número de problemas, en diferentes dominios, que pueden ser formulados mediante este tipo de grafos. Podemos encontrar estos problemas en la optimización de redes para arquitecturas de ordenadores paralelas, diseño de circutos VLSI, recuperación de información, análisis numéricos, biología computacional, teoría de grafos, planificación y arqueología. La etiquetación de los grafos más interesantes son los NP-duros y sus versiones NP-completas, pero, para la mayoría de las aplicaciones, encontrar una solución rápida y con coste óptimo es más que suficiente. Como consecuencia, los algoritmos de aproximación y heurísticas efectivas son las que se ponen en práctica.

A causa de su importancia, existe un gran número de resultados relacionados con el diseño de estos problemas. Aquí intentaremos presentar una visión general sobre el actual estado del arte con respecto a los problemas de etiquetado de grafos. Nuestra propuesta será: por un lado, presentar una vista global de los últimos resultados; por otro lado, implementar algunas de nuestras propuestas a algunos tipos de problemas. Existen otras fuentes de información con otros aspectos en el etiquetado de grafos, y en esta memoria también haremos referencia a ellos: Chinn et al., Chung, Monien y Sudborough, Díaz, Mohar y Poljak, Bezrukov, Lai y Williams y Raspaud et al.

El rumbo que tomará esta memoria es el siguiente: comenzaremos, en la sección 1.2, definiendo formalmente los problemas sobre el etiquetado de grafos y se mostrarán varios resultados obtenidos. Una visión histórica con motivos y aplicaciones en el estudio de estos problemas en la sección 3. En la sección 4 presentaremos los resultados de algunos tipos de algoritmos estudiados previamente. A continuación, en la sección 5, presentaremos los metodos metaheurísticos que nos han ayudado ha resolver los problemas que implementaremos. Finalmente, en las secciones 6 y 7, presentaremos la resolución de algunos problemas utilizando técnicas metaheurísticas y debatiendo los resultados.

1.2 Definiciones y observaciones

Empezaremos con nuestra definición de los problemas de etiquetado de grafos y después relacionaremos los conceptos. De esta forma podemos obtener una vista general del problema. Finalmente, presentaremos algunos resultados básicos de éstos. Algunos de los conceptos que se expondrán durante la memoria mantendrán una traducción literal del inglés, ya que esta memoria está integramente escrita en castellano.

La definición de la teoría de grafos que usaremos, en general, no discierne del estándar en la informática. Así pues, un grafo es finito, no direccional y sin ciclos. Además, un grafo G tiene un conjunto de vértices (o nodos) denotado por $V(G)$ y un conjunto de aristas denotado por $E(G)$. La notación uv representa una arista unidireccional $\{u, v\}$. El grado de un vértice u en un grafo G es denotado como $deg(u) = deg_G(u)$ y el grado máximo de G como $\Delta(G)$. El conjunto de adyacentes de un vértice u en G es denotado como $\Gamma(G) = \Gamma_G(u) = \{v \in V(G) : uv \in E(G)\}$.

Un *etiquetado lineal*, o simplemente *etiquetado*, de un grafo no dirigido $G = (V, E)$ con $n = |V|$ vértices es una función biyectiva $\varphi : V \to [n] = \{1, ..., n\}$. Un etiquetado puede llamarse también como una *ordenación lineal*, un *etiquetado lineal* o una *numeración* de los vértices de un grafo. Denotaremos como $\phi(G)$ el conjunto de todos los diseños de etiquetación de un grafo G.

Dado un etiquetado φ de un grafo $G = (V, E)$ y un número entero i, definimos el conjunto $L(i, \varphi, G) = \{u \in V : \varphi(i) \leq i\}$ y el conjunto $R(i, \varphi, G) = \{u \in V : \varphi(u) > i\}$. La *arista cortada* («*edge cut*») en la posición i de φ es definida como:

$$\theta(i, \varphi, G) = |\{uv \in E : u \in L(i, \varphi, G) \wedge v \in R(i, \varphi, G)\}|$$

y la *arista cortada modificada* («*modified edge cut*») en la posición i de φ como

$$\zeta(i, \varphi, G) = |\{uv \in E : u \in L(i, \varphi.G) \wedge v \in R(i, \varphi, G) \wedge \varphi(u) \neq i\}|$$

El *corte de vértices* («*vertex cut*») o *separación* en la posición φ como

$$\delta(i, \varphi, G) = |\{u \in L(i, \varphi, G) : \exists v \in R(i, \varphi, G) : uv \in E\}|$$

Dado un etiquetado φ de G y la arista $uv \in E$, el tamaño de uv en φ es

$$\lambda(uv, \varphi, G) = |\varphi(u) - \varphi(v)|$$

Una forma común de representar un etiquetado φ de un grafo G es alinear sus vértices en una línea horizontal, situando cada vértice u en la posición $\varphi(u)$, como se muestra en la figura 1.1. Esta representación gráfica nos da una idea, fácil de entender, de la anterior

definición: dibujando una línea vertical justo después de la posicón i y después la posición $i+1$, los vértices de la izquierda de la línea estarán contenidos en $L(i, \varphi, G)$ y los vértices a su derecha en $R(i, \varphi, G)$. Es fácil computar el corte $\theta(i, \varphi, G)$ contando el número de vértices a cada lado de la línea. El corte modificado $\zeta(i, \varphi, G)$ cuenta todos las aristas en $\theta(i, \varphi, G)$ excepto aquellas que tienen el vértice $\varphi^{-1}(i)$ como brecha. Es fácil de computar la separación $\delta(i, \varphi, G)$ contando el número de vértices a la izquierda de la línea vertical. Finalmente, el tamaño $\lambda(uv, \varphi, G)$ de una arista uv corresponde a la distancia natural entre dos puntos de corte en la imagen.

Dado un etiquetado φ de un grafo $G = (V, E)$, su *etiquetado inverso* es denotado por φ^R y su definición es $\varphi^u(i) = |V| - \varphi(u) + 1$ para todo $u \in V$.

Un *etiquetado de coste* es una función F que asocia a cada etiqueta φ de un grafo G a un número entero $F(\varphi, G)$. Dado F: el problema de optimización del etiquetado asociado con F consiste en determinar algún etiquetado $\varphi^* \in \Phi(G)$ de una entrada a un grafo G tal que

$$F(\varphi^*, G) = \min_{\varphi \in \Phi(G)} F(\varphi, G)$$

Para cualquier F en G, definimos $MINF(G) = min_{\varphi \in \Phi(G)} F(\varphi, G)$.

Los costes de los problemas estudiados están descritos a continuación, junto con sus etiquetados, y los escribiremos en inglés para poder hacer referencia a ellos en otras partes de la memoria:

- *Bandwidth* (*BANDWIDTH*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $BW(\varphi^*, G) = MINBW(G)$ dónde

$$BW(\varphi, G) = \max_{uv \in E}(uv, \varphi, G).$$

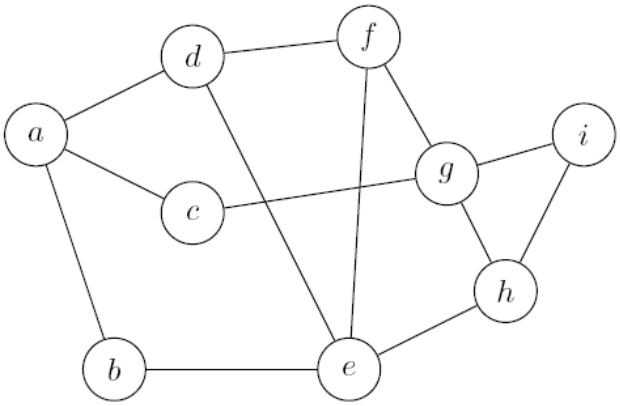
- *Minimum Linear Arrangement* (*MINLA*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $LA(\varphi^*, G) = MINLA(G)$ dónde

$$LA(\varphi, G) = \sum_{uv \in E} \lambda(uv, \varphi, G).$$

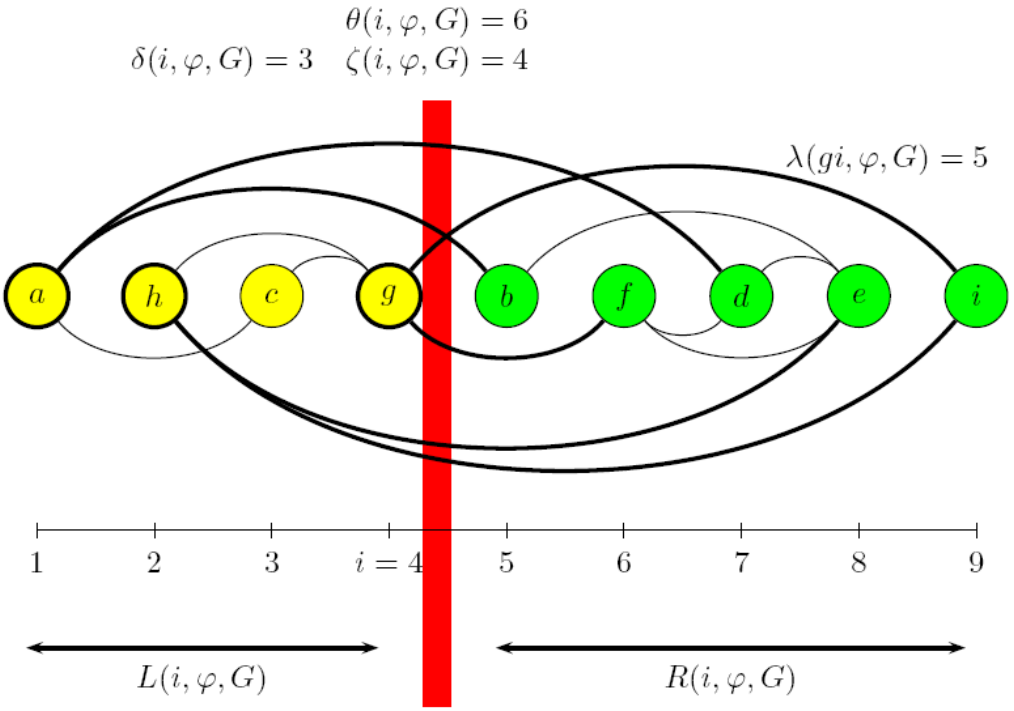
- *Cutwidth* (*CUTWIDTH)*: Dado un grafo $G = (V, E)$, econtrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $CW(\varphi^* = MINCW(G)$ dónde

$$CW(\varphi, G) = \max_{i \in [|V|]} \theta(i, \varphi, G).$$

- *Modified Cut* (*MODCUT*): Dado un grafo $G = (V, E)$, encontrar un etiquetado



(a) $G = (V, E)$



(b) Representación gráfica de φ. Dividiendo el diseño en $i = 4$ los vértices de la izquierda son mostrados en amarillo, los vértices de la derecha en verde oscuro, las aristas cortadas y el separador estan destacados.

Figura 1.1: Un grafo G junto con algunos diseños y una representación gráfica del etiquetado $\varphi = \{(a, 1), (b, 5), (c, 3), (d, 7), (e, 8), (f, 6), (g, 4), (j, 9), (h, 2)\}$.

$\varphi^* \in \Phi(G)$ tal que $MC(\varphi^*, G) = MINMC(G)$ dónde

$$MC(\varphi, G) = \sum_{i \in [|V|]} \zeta(i, \varphi, G).$$

- *Vertex Separation* (*VERTSEP*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $VS(\varphi^*, G) = MINVS(G)$ dónde

$$VS(\varphi, G) = \max_{\imath \in [|V|]} \delta(i, \varphi, G).$$

- *Sum Cut* (*SUMCUT*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $SC(\varphi^*, G) = MINSC(G)$ dónde

$$SC(\varphi, G) = \sum_{i \in [|V|]} \delta(i, \varphi, G).$$

- *Profile* (*PROFILE*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $varphi^* \in \Phi(G)$ tal que $PR(\varphi^*, G) = MINPR(G)$ dónde

$$PR(\varphi, G) = \sum_{u \in V} (\varphi(u) - \min_{v \in \Gamma^*(u)} \varphi(v)).$$

  y $\Gamma^*(u) = \{u\} \cup \{v \in V : uv \in E\}$.

- *Edge Bisection* (*EDGEBIS*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $EB(\varphi^*, G) = MINEB(G)$ dónde

$$EB(\varphi, G) = \theta(\left\lfloor \frac{1}{2}|V| \right\rfloor, \varphi, G).$$

- *Vertex Bisection* (*VERTBIS*): Dado un grafo $G = (V, E)$, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $VB(\varphi^*, G) = MINVB(G)$ dónde

$$VB(\varphi, G) = \delta(\left\lfloor \frac{1}{2}|V| \right\rfloor, \varphi, G).$$

Estrictamente hablando, los problemas *EDGEBIS* y *VERTBIS* no son problemas de etiquetado: ambos problemas buscan una partición en el conjunto de vértices del gráfico en dos subconjuntos del mismo tamaño (o difiriendo por un número de vértices sin pareja). No obstante, problemas de bisección y de etiquetado están estrechamente relacionados.

En este punto de la memoria es relevante señalar algunos, básicos, pero importantes hechos.

1. Para un grafo $G = (V, E)$ y un etiquetado φ de G, el tamaño total de sus aristas es igual a la suma de todas aristas cortadas en el diseño:

$$sum_{uv \in E} \lambda(uv, \varphi, G) = \sum_{i \in [|V|]} \theta(i, \varphi, G).$$

   Este hecho fue anunciado por Harper en 1966. Continuando desde la observación de que un arista $uv \in E$ con $\varphi(u) < \varphi(v)$ contribuye $\varphi(v) - \varphi(u)$ a la izquierda y 1 a cada uno de los términos $\theta(\varphi(u), \varphi, G), \theta(\varphi(u) + 1, \varphi, G), ..., \theta(\varphi(v), \varphi, G)$ en la parte derecha.

2. Para un grafo $G = (V, E)$ y un etiquetado φ de G,

$$PR(\varphi, G) = SC(\varphi^R, G).$$

   Esta identidad ha sido reconocida aparentemente durante un tiempo; además una reciente prueba puede ser encontrada en el artículo de Golovach y Fomin de 1998. Ésto es debido al hecho de que cada vértice $u \in V$ contribuye una unidad $\varphi(u) - min_{v \in \Gamma^*(u)} \varphi(v)$ a la suma cortada («*sum cut*») en el etiquetado inverso. Así pues, debido a ésto, *PROFILE* y *SUMCUT* son problemas equivalentes.

3. Es importante señalar que los problemas de etiquetado de grafos han sido formulados explicitamente para solicitar la construcción de un etiquetado con un coste óptimo mas que para que la construcción sea óptima:

   «*Dado un grafo G, encontrar un etiquetado $\varphi^* \in \Phi(G)$ tal que $F(\varphi^*, G) = MINF(G)$.*»

   Todos los problemas pueden, sin embargo, volverse a formular como problemas de decisión, donde la tarea es decidir si admite un grafo o no un etiquetado con coste no tan bueno como un entero dado como parte de su entrada:

   «*Dado un grafo G y un entero k, ¿hay algún etiquetado $\varphi \in \Phi(G)$ tal que $F(\varphi, G) \leq k$?*».

   El siguiente teorema nos da algunas relaciones entre el tamaño de los costes de los etiquetados.

Lema 1 *Siendo G cualquier grafo con n vértices y m aristas, y siendo φ cualquier etiquetado de G. Entonces:*

$$
\begin{array}{ll}
LA(\varphi,G) \leq n \cdot CW(\varphi,G), & MINLA(G) \leq n \cdot MINCW(G), \\
LA(\varphi,G) \leq m \cdot CW(\varphi,G), & MINLA(G) \leq m \cdot MINCW(G), \\
MC(\varphi,G) \leq LA(\varphi,G), & MINMC(G) \leq MINLA(G), \\
SC(\varphi,G) \leq n \cdot VS(\varphi,G), & MINSC(G) \leq n \cdot MINVS(G), \\
VS(\varphi,G) \leq BW(\varphi,G), & MINVS(G) \leq MINBW(G), \\
EB(\varphi,G) \leq CW(\varphi,G), & MINEB(G) \leq MINCW(G), \\
VB(\varphi,G) \leq VS(\varphi,G), & MINVB(G) \leq MINVS(G), \\
CW(\varphi,G) \leq \Delta(G) \cdot BW(\varphi,G), & MINCW(G) \leq \Delta(G) \cdot MINBW(G)
\end{array}
$$

El siguiente lema nos muestra los costes de algunos etiquetados de un grafo en terminos de su conectividad con sus componentes. Un estudio para el bando de ancha («*bandwidth*») puede ser encontrado en el artículo de Chvátalvá et al. de 1975; para los problemas restantes, los resultados pueden ser probados de forma similar.

Lema 2 *Siendo G un grafo y $G_1, ..., G_k$ sus componentes conectados. Entonces,*

$$
\begin{array}{ll}
MINBW(G) = \max\limits_{i \in [k]} MINBW(G_i), & MINMC(G) = \sum_{i \in [k]} MINMC(G_i), \\
MINCW(G) = \max\limits_{i \in [k]} MINCW(G_i), & MINLA(G) = \sum_{i \in [k]} MINLA(G_i), \\
MINVS(G) = \max\limits_{i \in [k]} MINVS(G_i), & MINSC(G) = \sum_{i \in [k]} MINSC(G_i)
\end{array}
$$

Una útil consecuencia del anterior lema es que, para cada etiquetado de coste $F \in \{BW, CW, VS, LA, MC, SC\}$, es posible obtener un etiquetado óptimo para F en un grafo computacionando el etiquetado óptimo de sus componentes conectados. Observamos, sin embargo, que los costes de las bisecciones EB y VB no comparten esta propiedad: como contraejemplo, consideraremos un grafo creado con dos componentes del mismo tamaño.

El siguiente lema está relacionado con el coste del etiquetado de un grafo y su subgrafo. Dado dos grafos G y H, entonces H es una arista inducida en el grafo G si $V(H) = V(G)$ y $E(H) \subseteq E(G)$ y también si H es un vértice inducido en el grafo de G si $V(H) \subseteq V(G)$ y $E(H) = \{uv \in E(G) : u, v \in V(H)\}$.

Lema 3 *Siendo H una arista o un vértice inducido en el subgrafo de un grafo G. Entonces, para el coste de etiquetado $F \in \{BW, CW, VS, LA, MC, SC\}$, mantiene que $MINF(H) \leq MINF(G)$. En el caso de que H sea una arista inducida en el subgrafo de G, entonces $MINEB(H) \leq MINEB(G)$ y $MINVB(H) \leq MINVG(G)$.*

1.3 Herramientas utilizadas

Durante el desarrollo de este proyecto se implementaron los programas mediante C♯ con .NET Framework utilizando *Visual Studio 2005*.

1.3.1 Visual Studio 2005

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C♯, Visual J♯, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión NET 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Visual Studio 2005 se empezó a comercializar a través de Internet a partir del 4 de Octubre de 2005 y llegó a los comercios a finales del mes de Octubre en inglés. En castellano no salió hasta el 4 de Febrero de 2006. Microsoft eliminó .NET, pero eso no indica que se alejara de la plataforma .NET, de la cual se incluyó la versión 2.0 de la máquina virtual Java.

La actualización más importante que recibieron los lenguajes de programación fue la inclusión de tipos genéricos, similares en muchos aspectos a las plantillas de C♯. Con esto se consigue encontrar muchos más errores en la compilación en vez de en tiempo de ejecución, incitando a usar comprobaciones estrictas en áreas donde antes no era posible. C++ tiene una actualización similar con la adición de C++/CLI como sustituto de C♯ manejado.

Se incluye un diseñador de implantación, que permite que el diseño de la aplicación sea validado antes de su implantación. También se incluye un entorno para publicación web y pruebas de carga para comprobar el rendimiento de los programas bajo varias condiciones de carga.

Visual Studio 2005 también añade soporte de 64-bit. Aunque el entorno de desarrollo sigue siendo una aplicación de 32 bits Visual C++ 2005 soporta compilación para x86-64 (AMD64 e Intel 64) e IA-64 (Itanium). El SDK incluye compiladores de 64 bits así como versiones de 64 bits de las librerías.

1.3.2 El lenguaje de programación C♯

C♯ es un lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también

conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C♯ es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C♯ es más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C♯ carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C♯ es el lenguaje nativo de .NET.

La sintaxis y estructuración de C♯ es muy parecida a la de C++ o Java, puesto que la intención de Microsoft es facilitar la migración de códigos escritos en estos lenguajes a C♯ y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son comparables con los de Visual Basic.

En resumen, C♯ es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el .NET Framework SDK.

1.3.3 .NET Framework

Microsoft .NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada plataforma .NET, y a los servicios antes comentados se les denomina servicios Web.

Para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), Microsoft ha publicado el denominado kit de desarrollo de software conocido como .NET Framework SDK, que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución y Visual Studio.NET, que permite hacer todo la anterior desde una interfaz visual basada en ventanas.

El concepto de Microsoft.NET también incluye al conjunto de nuevas aplicaciones que Microsoft y terceros han (o están) desarrollando para ser utilizadas en la plataforma .NET. Entre ellas podemos destacar aplicaciones desarrolladas por Microsoft tales como Windows.NET, Hailstorm, Visual Studio.NET, MSN.NET, Office.NET, y los nuevos servidores para empresas de Microsoft (SQL Server.NET, Exchange.NET, etc.).

1.3.4 Common Languaje Runtime

El *Common Language Runtime* (CLR) es el núcleo de la plataforma .NET. Es el motor encargado de gestionar la ejecución de las aplicaciones para ella desarrolladas y a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad. Posee de un modelo de programación consistente y sencillo, es decir, es un modelo de programación orientado a objetos, en lo que se diferencia de algunas librerías DLL que ofrecen funciones globales. La programación con .NET Framework abstrae al programador de conceptos de sistemas operativos como el registro de Windows, GUIDs, HRESULTS, IUnknown, etc.

Una de las nuevas ventajas de esta plataforma es la desaparición del concepto conocido como «*infierno de las DLL*» en las que librerías DLL deben ser sustituidas por nuevas versiones. A su vez, ciertos programas que comparten estas librerías en común pueden tener problemas de convivencia. En .NET las nuevas versiones de las DLLs pueden coexistir con las viejas.

El CLR actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows. Asimismo, Microsoft está acordando portar las librerías a otros sistemas como Linux. Además, desde cualquier lenguaje para el que exista un compilador que genere código para la plataforma .NET es posible utilizar código generado para la misma usando cualquier otro lenguaje tal y como si de código escrito usando el primero se tratase.

El CLR incluye un recolector de basura que evita que el programador tenga que tener en cuenta cuándo ha de destruir los objetos que dejen de serle útiles. Este recolector es una aplicación que se activa cuando se quiere crear algún objeto nuevo y se detecta que no queda memoria libre para hacerlo. En ese caso, buscará en la memoria dinámica de la aplicación para liberar objetos en desuso.

El CLR detecta el correcto tipado de los objetos, con esto, a parte de ayudar al programador durante la compilación, aseguramos que los procesos no puedan acceder a código de otros puesto que no pertenecen al mismo tipo. De igual forma, podemos controlar de manera efectiva las excepciones.

El CLR permite que excepciones lanzadas desde código para .NET escrito en un cierto lenguaje se puedan capturar en código escrito usando otro lenguaje, e incluye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro.

Otra de las ventajas del CLR es la interoperabilidad con código antiguo, es decir, desde el código escrito para la plataforma .NET podemos tener acceso a objetos COM.

1.3.5 Microsoft Intermediate Languaje

Ninguno de los compiladores para la plataforma .NET produce código máquina para la CPU. Estos compiladores generan un lenguaje intermedio llamado MSIL (*Microsoft Intermediate Languaje*) que puede ser interpretado por el CLR, como podemos ver en la figura 1.2.

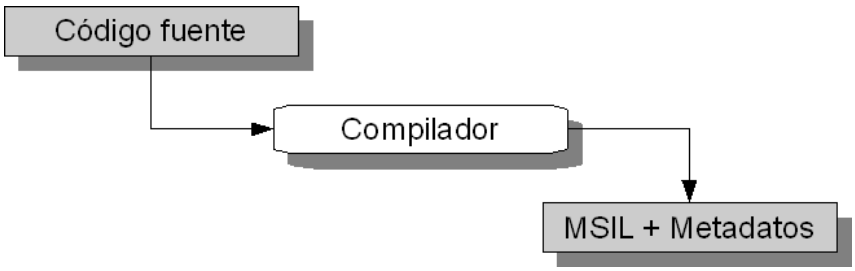


Figura 1.2: Creación de código MSIL a partir del código fuente.

La principal ventaja del MSIL es la facilitación de la ejecución multiplataforma, ya que se pueden crear CLR para diferentes sistemas sin cambiar el código inicial. Sin embargo, puesto que la CPU no puede ejecutar código directamente del MSIL, se deberá convertir a código nativo de la CPU. Para ello, el CLR tiene un componente llamado *jitter* (*Just-In-Time*) que se encarga de esta tarea. En diferencia a lenguajes como Java, el *jitter* compilará a código nativo una única vez durante el proceso de la ejecución del programa y no interpretará constantemente el código del lenguaje intermedio cuando lo necesita. Podemos ver el proceso del *jitter* en la figura 1.3.

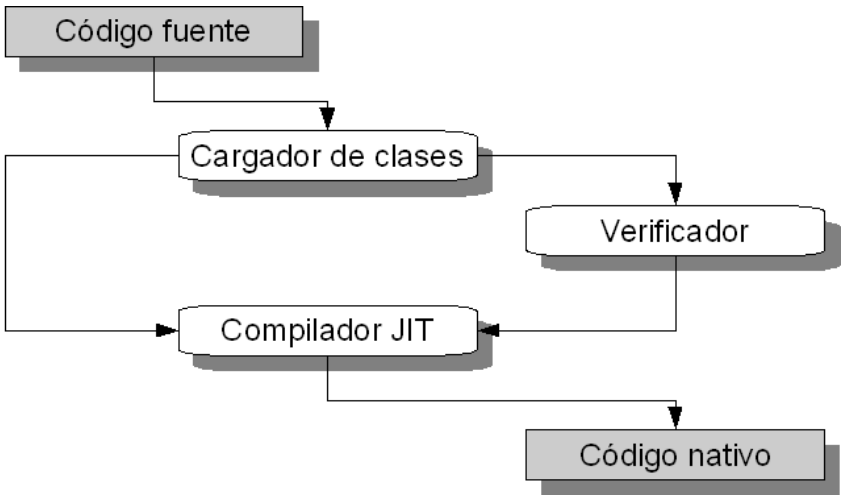


Figura 1.3: Conversión del código MSIL a código nativo.

1.3.6 Ensamblados

Un ensamblado es una agrupación lógica de uno o más módulos o ficheros de recursos (ficheros .GIF, .HTML, etc.) que se engloban bajo un nombre común. Un programa puede acceder a información o código almacenados en un ensamblado sin tener que conocer cuál es el fichero en concreto donde se encuentran, por lo que los ensamblados nos permiten abstraernos de la ubicación física del código que ejecutemos o de los recursos que usemos. Por ejemplo, podemos incluir todos los tipos de una aplicación en un mismo ensamblado pero colocando los más frecuentemente usados en un cierto módulo y los menos usados en otro, de modo que sólo se descarguen de Internet los últimos si es que se van a usar.

2. OBJETIVOS

El objetivo principal de este proyecto será la aplicación de metaheurísticas a problemas reales de etiquetación de grafos. Además, intentar proporcionar resultados que se aproximen al estado del arte actual y hacer una estudio y revisión de estos resultados que nos proporcionen un conocimiento apliado en este dominio. Para alcanzar esta meta, es necesario alcanzar los siguientes objetivos operativos:

- Objetivos de implementación:
  - Aprender a utilizar el entorno de desarrollo integrado Visual Studio 2005.
  - Familiarizarse con el lenguaje de programación C♯.
  - Conocer el marco de trabajo .NET, así como las librerías necesarias básicas que nos ayuden a implementar el objetivo principal.
- Objetivos sobre los problemas a tratar:
  - Conocer en detalle los problemas a tratar: definición descriptiva y formal.
    1. El problema del etiquetado lineal mínimo.
    2. El problema del perfil.
  - Realizar un estudio sobre la historia y el estado del arte de estos problemas y similares.
- Objetivos sobre la implementación de un algoritmo que nos ayude a obtener una solución óptima:
  - Hacer un estudio de las metaheurísticas existentes.
  - Adaptar los algoritmos heurísticos a los problemas que queremos resolver.

3. PERSPECTIVA HISTÓRICA

Así como mencionaremos, el término *etiquetado* y *problema de etiquetado* son antiguos problemas para resolver el etiquetado óptimo de circuitos. Aquí mostraremos algunos transfondos que nos motivarán en la investigación de problemas del etiquetado, también algunas de sus aplicaciones. Empezaremos con una visión histórica general.

El problema del etiquetado mínimo lineal (MINLA) fue citado por Harper en 1964. A Harper le mandaron diseñar un etiquetado de códigos error-corrección con la mínima media de errores absolutos para algunos tipos de grafos. Después, este problema fue también considerado por Mitchison y Durbin en 1986 para realizar un sobre-simplificado modelo de la actividad nerviosa del cortex. MINLA también fue aplicado en máquinas personales para la planificación, según Adolphson en 1977 y Ravi et al. en 1991, y el dibujo de grafos, según Shahrokhi et al. en 2001. El problema del etiquetado mínimo lineal ha recibido diversos tipos de nombres, como (en inglés) *Minimum linear Arrangement Problem*, *Optimal linear Ordening*, *Edge Sum*, *Minimum-1-sum*, *Bandwidth Sum* o *Wirelength Problem*.

El Ancho de Banda («*Bandwidth*») recibió mucha atención durante los años cincuenta con el fin de acelerar la computación de matrices. Según Dewdney en 1976, la introducción del problema del ancho de banda para los grafos fue nombrado por Harary en 1967, sin embargo el problema fue definido formalmente por Harper en 1966.

El problema del Ancho de Corte (*Cutwidth*) fue inicialmente usado en los años setenta como un modelo teórico para numerar los canales en un etiquetado óptimo de circuitos, descrito por Adolphson y Hu en 1973; se puede encontrar también en la introducción del artículo de Makedon y Sudborough en 1989. Las más recientes aplicaciones de este problema incluyen la fiabilidad en las redes, descritas por Karger en 1999, el dibujo automático de grafos, por Mutzel en 1995, y en la recuperación de información, por Botafogo en 1993.

El problema de Separación de Vértices («*Vertex Separation Problem*», *VERTSEP*) fue propuesto en un principio para resolver el problema general de buscar buenas formas de separar grafos, citado por Lipton y Tarjan en 1979, y sus aplicaciones en algoritmos para los etiquetados VLSI, descrito por Leiserson en 1980. Así pues veremos este problema como un problema equivalente a otros ya conocidos.

Los problemas del Corte de Suma (*Sum Cut*, *SUMCUT*) y del Perfil (*Profile*, *PROFILE*) en grafos son indirectamente nombrados en el artículo de Díaz et al. en 1991 y en el artículo de Lin y Yuan en 1994. El problema *SUMCUT* fue originalmente propuesto como una simplificación de la versión del problema del operador-φ. El problema del *PROFILE* fue propuesto para reducir la cantidad de espacio en las matrices, descrito por los

autores Tewarson, Lepin, Lin y Yuan. Ambos problemas tienden a ser equivalentes con el problema del Grafo con Intervalo de Finalización (*Interval Graph Completion*) de Ravi et al. en 1991, el cual tiene aplicaciones en arqueología, descubiertas por Kendall en 1969, y en la clonación de huellas dactilares de Karp en 1993. El problema fue redescubierto otra vez por Golovach en 1997 bajo el nombre de *número de separaciones totales de vértices* (*total vertex separation number*).

El problema de la Bisección de Aristas (*Edge Bisection*, *EDGEBIS*) tiene un vasto rango de aplicaciones, notablemente en el área de la computación paralela y VLSI, citado por varios autores como Bhatt y Leighton en 1984, Shing y Hu en 1986, Hromkovic y Monien en 1992, Leighton en 1993 y Diekmann et al. en 1994. El problema *EDGEBIS* es relevante en la tolerancia a fallos y está emparentado con la complejidad a la hora de mandar mensajes a los procesadores en las redes de interconexión vía caminos de vértices separados, propuesto por Klasing en 1998.

En el resto de esta sección, presentaremos algunas aplicaciones lejanas emparentadas con este tipo de problemas de etiquetado.

3.1 Problemas de etiquetado en el análisis numérico

En el área del análisis numérico, es deseabe para muchos programadores reordenar columnas y filas de muchas matrices de forma que sus entradas no-cero se encuentren lo más cerca de la diagonal. El ancho de banda de una matriz simétrica M es el mayor entero B para el cual hay una entrada no-cero en $M[j, j+b]$ y el $perfil$ de M es $\sum_{i\in[n]}(i-p_i)$ dónde p_j es el índice de la primera entrada no-cero de la fila j. Reduciendo el ancho de banda y/o el perfil de una matriz, ésta tiende a reducir la cantidad de espacio necesario para almacenar algunos esquemas de almacenamiento y además a mejorar el rendimiento de algunas operaciones comunes como en la factorización de Cholesky de un sistema no-singular de ecuaciones. El problema de reducir el ancho de banda o el perfil de una matriz M consiste en encontrar una permutación P de la matriz tal que $M' = P \cdot M \cdot P^T$ con el mínimo ancho de banda o mínimo perfil. Retomando ésto, una permutación P de la matriz es una matriz identidad con el mismo tamaño de M en las cuales sus columnas han sido permutadas.

Observamos que si identificamos las entradas no-cero de una matriz simétrica con las aristas de un grafo y las permutuaciones de sus filas y columnas con cambios en las etiquetas de los vértices, entonces el ancho de banda del grafo equivale al ancho de banda de la matriz, y el perfil de un grafo equivale al perfil de la matriz.

El problema de la reducción del ancho de banda o el perfil de una matriz simétrica posee una larga historia que comenzó en los años cincuenta; se puede ver un ejemplo en la referencia de Gibbs et al. de 1976. Actualmente, existen métodos generales que son más eficientes que los «*envelope schemes*». No obstante, muchos paquetes comerciales todavía

ofrecen funciones para reducir el ancho de banda de matrices como un método de preprocesado. De este modo, las mejoras en estos métodos pueden ser portadas al programa sin una completa reorganización de su arquitectura. Algoritmos eficientes desempeñan varias operaciones en matrices en las que pequeños anchos de banda pueden ser encontrados. La recuperación de información para navegar por el hipertexto es una área reciente dónde también se usan las técnicas de reducción del ancho de banda y de perfil.

3.2 Problemas de etiquetado en VLSI

Muchos problemas de etiquetado fueron originalmente motivados para la simplificación de modelos matemáticos del etiquetado de VLSI. Dado un conjunto de módulos, el problema del etiquetado de VLSI consiste en reemplazar los módulos en un circuto de forma que no se sobrepongan y cablear las terminales junto con los diferentes módulos de acuerdo a la especificación del cableado de forma que los cables no interfieran entre ellos. Hay dos fases en el etiquetado de VLSI: asignación y enrutado. El *problema de la asignación* consiste en colocar los módulos en el circuito; el *problema del enrutado* consiste en cablear desde las terminales de diferentes módulos para que puedan estar interconectados. Un circuito VLSI puede ser modelado como un grafo, donde las aristas representan los cables y los vértices representan los módulos. De esta forma, el grafo es un modelo simplificado del circuito, así pues sobreentendiendo y encontrando problemas de este simple modelo puede ayudarnos a obtener mejores soluciones para el modelo real.

Un primer estudio para resolver la fase de asignación fue usado en el problema del Etiquetado Lineal Mínimo (*Minimum Linear Arrangement*) de forma que minimizamos la longitud del cableado. Recientemente se ha citado un estudio alternativo para resolver este problema. Consiste en encontrar recursivamente cortes mínimos con la mínima capacidad entre todos los cortes que separan el grafo en dos componentes de igual tamaño. En el estudo de Simon y Teng en 1997 fue propuesto el problema de las bisección de aristas (*Edge Bisection*). En la actualidad, la tecnología de los circuitos integrados ha cambiado sustancialmente y algunas de las primeras aplicaciones de etiquetados están obsoletas.

El ancho de corte (*CUTWIDTH*) de un grafo multiplica el orden de un grafo dado en medida del área necesitada para representar el grafo en un diseño VLSI cuando los vértices son establecidos en una fila. De hecho, Raspaud et al. en 1995 provaron una nueva relación entre el ancho de corte y el área del etiquetado VLSI de un grafo: el área mínimo de un etiquetado VLSI de un grafo no es menor que un cuadrado de su ancho de corte. Para gráficos con el máximo grado de cuatro se encontró una relación similar entre el área y la bisección de aristas.

3.3 Problemas de etiquetado en el dibujo de grafos

Quizas, uno de las metas más importantes en el dibujo de grafos es reducir la área en la que se representa un grafo. Reduciendo el número de aristas cruzadas es la mejor forma de mejorar la lectura y la comprensión de un grafo. Un dibujo bipartito o un dibujo de dos capas es la representación de un grafo dónde los vértices de un grafo bipartito son reemplazados en dos líneas paraleas y las aristas son dibujadas con líneas rectas entre ellas. El número de cruces bipartitos de un grafo bipartito es el mínimo número de aristas que cruzan sobre ellas. Shahrokhi et al. en 2001 probaron que para una gran clase de grafos bipartitos, reducir el número de cruces bipartitos es equivalente a reducir el total de la longitud de sus aristas, esto es, el Etiquetado linear Mínimo. Además, una solución aproximada al *MINLA* puede ser usada para generar una solución aproximada al problema del Número de Cruces Bipartitos y en el artículo de Vrto de 2002 hay una extensa bibliografía sobre ésto.

3.4 Problemas de etiquetado en procesamiento paralelo y distribuido

Muchos ordenadores en paralelo están compuestos por un conjunto de procesadores con su propia memoria privada que intercambia mensajes por la red de comunicación. Con el fin de establecer mejor velocidad cuando usamos un sistema como éste, es importante distribuir la cantidad de trabajo entre procesadores. También es importante reducir la cantidad de comunicación entre los procesadores, porque la comunicación a través de la red es mucho más lenta que la velocidad de los procesadores.

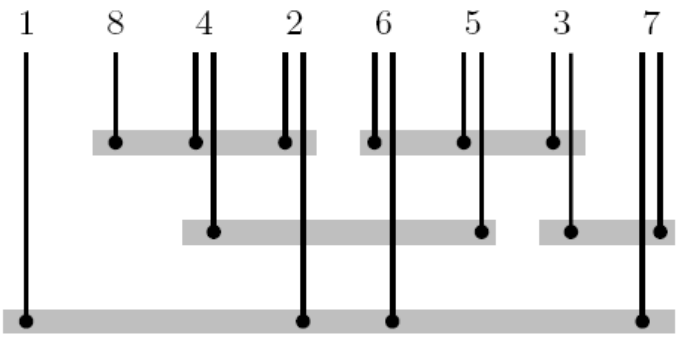
El problema de la Partición del Grafo consiste en particionar los vértices de un grafo dado en k conjuntos con parecido tamaño de forma que el número de vértices cortados entre los k conjuntos sea el mínimo. El problema de la bisección de las aristas es un caso particular del Particionado del Grafo dónde $k = 2$. La bisección recursiva es una técnica popular para obtener particiones cuando k es 2. En los artículos de Simon y Teng de 1997 y en las referencias de Bezrukov de su artículo de 1999 podemos ver con más detalle un análisis de la bisección recursiva y sobre el particionamiento de un grafo.

La Bisección de las Aristas puede ser usada cuando queremos resolver ecuaciones parciales diferenciales. Simplificando, en estos problemas una tarea interactiva computacional debe ser llevada a cabo desde cada vértice de una maya (o gráfico definido por una particular topología del sistema) y su computación supone usar los datos de sus vértices y sus adyacentes. Un camino para distribuir la cantidad de computación entre dos procesadores es asignar a cada uno la mitad de los vértices en la maya. Pero como las fronteras de los vértices necesitan comunicarse, es necesario reducir el corte de la bisección.

3.5 Algunos problemas equivalentes

El problema de la Separación de Vértices está estrechamente vinculado con otros importantes problemas NP-completos: *Gate Matrix Layout*, *Pathwidth* y *Vertex Search Number*. El problema del Ancho de Banda es también pariente del *Proper Pathwidth*.

El problema del Etiquetado de Puertas de una Matriz («*Gate Matrix Layout*») es un problema ya estudiado para aplicaciones en el etiquetado de circuitos CMOS. Un ejemplo del problema del «*Gate Matrix Layout*» consiste en una colección de redes $\{G_1, ..., G_m\}$. Las redes son identificadas con filas y las redes son identificadas con columnas. La meta de este problema es encontrar una permutación de columnas que minimicen el número de pistas requeridas para construir el chip, lo cual es equivalente a minimizar su área. La figura 3.1 muestra un ejemplo de *Gate Matrix Layout*. Expresaremos con $MINGML(G)$ el mínimo número de pistas necesarias por un grafo G.



	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
N_1	1	1	0	0	0	1	1	0
N_2	0	0	0	1	1	0	0	0
N_3	0	0	1	0	0	0	1	0
N_4	0	1	0	1	0	0	0	1
N_5	0	0	1	0	1	1	0	0

Figura 3.1: Ejemplo de una *Gate Matrix Layout* con tres pistas

El problema de la Anchura del Camino («*Pathwidth*») ha tenido un gran interés en estos años por su relación con la teoría de grafos secundarios citados por Robertson y Seymour en 1985. Una descomposición del camino de un grafo $G = (V, E)$ es una secuencia de subconjuntos de vértices $(X_1, ..., X_r)$ tal que:

- $\bigcup_{i=1}^{r} X_i = V$;
- ambos extremos de cualquier arista $e \in E$ pertenece a algún X_i para $1 \leq i \leq r$; y

- para todo $i \leq j \leq k$, se cumple $X_i \cap X_k \subseteq X_j$.

El *pathwidth* de un camino de descomposición $(X_1, ..., X_r)$ es

$$MINPW(G, (X_1, ..., X_r)) = \max_{i \in [r]} |X_i| - 1.$$

El *pathwidth* de G, expresado por $MINPW(G)$, es el mínimo *pathwidth* sobre todos los posibles caminos de descomposición de G. El problema *PATHWIDTH* consiste en determinar un camino de descomposición con el mínimo *pathwidth*. La figura 3.2 muestra un gráfico con uno de sus caminos de descomposición en el problema *PATHWIDTH*.

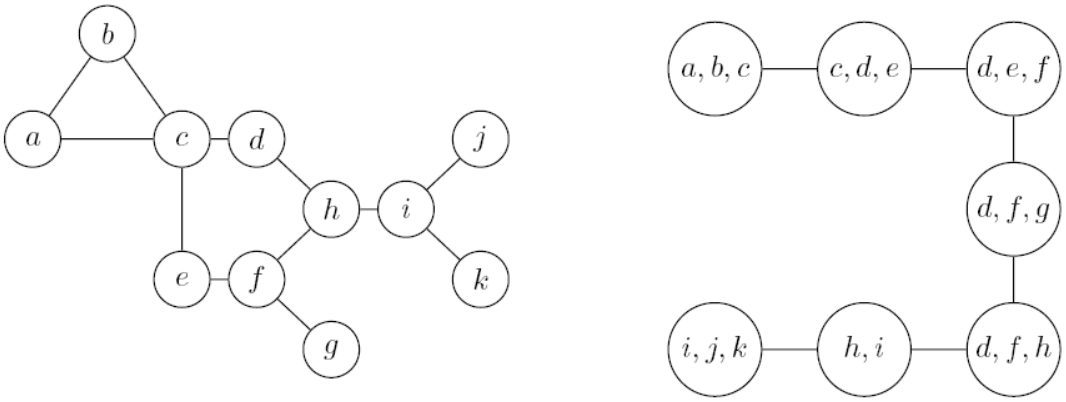


Figura 3.2: Ejemplo de un grafo y uno de sus caminos en descomposición con *pathwidth* 2.

El problema de la Búsqueda del Número de Vértices («*Vertex Search Number*») fue nombrado por Kirousis y Papadimitriou en 1986. Al poco tiempo, muchos investigadores se preguntaban si tenían la necesidad de capturar un ilimitado número de intrusos moviéndose a través de las aristas de un grafo dado. Denotaremos como $MINSN(G)$ la *minimal vertex search* de un grafo G.

La equivalencia entre los problemas *Gate Matrix Layout*, *Search Number*, *Pathwidth* y *Vertex Separation* es una consecuencia de los resultados de Kirousis y Papadimitriou en 1986; Kinnersley en 1992; Fellows y Langston en 1994:

Teorema 1 *Para cualquier grafo G,*

$$MINVS(G) = MINPW(G) = MINSN(G) - 1 = MINGML(G) + 1.$$

Siendo $I_v = \{i : v \in X_i\}$. Un *apropiado camino de descomposición* es un camino de descomposición que también satisface $I_u \not\subseteq I_v$ para todo $u, v \in V$. El apropiado *pathwidth* de G, denotado como $MINPPW(G)$, es el mínimo *pathwidth* sobre todos los apropiados caminos de descomposición de G. El problema del *Apropiado Pathwidth* consiste en determinar un apropiado camino de descomposición con el mínimo *pathwidth*.

La siguiente equivalencia entre el Ancho de Banda y el Apropiado *Pathwidth* se debe a Kaplan y Shamir en 1996:

Teorema 2 *Para cualquier grafo* G, $MINPPW(G) = MINBW(G)$.

4. ESTUDIOS PREVIOS

4.1 Resultados NP-Completos

Normalmente decimos que un problema es NP-Completo si es un problema intratable computacionalmente. El siguiente teorema indica la dificultad de los problemas de etiquetado en grafos arbitrarios.

Teorema 3 *Los problemas de etiquetado BANDWIDTH, MINLA, CUTWIDTH, MODCUT, VERTSEP, SUMCUT y EDGEBIS son NP-Completos.*

Estas deduciones fueron dadas por los siguientes autores: el *BANDWIDTH* por Papadimitriou en 1976; para *MINLA* y *EDGEBIS* por Gare et al. en 1976; para el *CUTWIDTH* por Gavril en 1988; para *MODCUT* por Moniem y Subdorough en 1988; para *VERTSEP* por Lengauer en 1981; y para el *SUMCUT* fue encontrado por Díaz et al. en 1991, Lin y Yuan en 1994 y por Golovach en 1997.

Muchos de los problemas de etiquetado son NP-Completos para algunos tipos de grafos, la tabla 4.1 muestra una visión general del estudio de estos tipos de grafos.

La complejidad sigue sin ser calculada para algunos de estos problemas de etiquetado.

4.2 Tipos de grafos con algoritmos de tiempo polinómico

Los resultados NP-Completos no nos enseñan a aplicar algoritmos eficientes para obtener los resultados óptimos para algunos particulares tipos de grafos. En esta sección revisaremos algunos resultados para problemas de etiquetado.

En el caso del problema del etiquetado mínimo (*MINLA*), el valor óptimo para hipercubos es conocido y existe una fórmula para obtener el valor del grafo Bruijn de orden cuatro. La motivación para realizar este estudio fue minimizar el total de la longitud de las aristas para la conexión de los cables del decodificador Viterbi y la motivación posterior fue para diseñar los códios error-conexión con el mínimo número de errores. En el caso de un hipercubo de d-dimensiones Q_d, $MINLA(Q_d) = 2^{d-1}(2_d - 1)$.

Chung, Goldberg y Klipker fueron los primeros en obtener un algoritmo de complejidad $O(n^3)$ para resolver el problema del Etiquetado Lineal Mínimo para árboles. Adolphson y Hu obtenieron un algoritmo de complejidad $O(nlogn)$ para computar el problema *MINLA* con un árbol con n raices. Shiloach mejoró los resultados presentando un algoritmo para el *MINLA* para árboles de n vértices con complejidad $O(n^{2,1})$ en tiempo. El valor óptimo

Problema	NP-Completo	Autor
BANDWIDTH	en general	Papadimitriou, 1976
	para árboles con máximo grado 3	Garey et al, 1978
	para orugas con tamaño de pelo ≤ 3	Monien, 1986
	para orugas con pelo ≤ 1 por vértices de columna vertebral	Monien, 1986
	para orugas ciclicas con pelo 1	Muradyan, 1999
	para grafos de mallas y grafos de unidad de disco	Díaz et al, 2001
MINLA	en general	Garey et al, 1976
	para grafos bipartitos	Even y Shiloach, 1975
CUTWIDTH	en general	Gavril, 1977
	para grafos con máximo grado 3	Makedon et al. 1985
	para grafos de dos dimensiones con máximo grado 3	Monien y Sudboriugh, 1988
	para grafos de mallas y grafos de unidad de disco	Díaz et al, 2001
MODCUT	para grafos de dos dimensiones con máximo grado 3	Monien y Sudborough, 1988
VERTSEP	en general	Lengauer, 1981
	para grafos de dos dimensiones con máximo grado 3	Monien y Sudborough, 1988
	para grafos de cuerdas	Gustedt, 1993
	para grafos bipartitos	Goldberg et al, 1995
	para grafos de mallas y grafos de unidad de disco	Díaz et al, 2001
SUMCUT	en general	Díaz et al, 1991
		Lin y Yuan, 1994
		Golovach, 1997
	para grafos cobipartitos	Yuan et al, 1987
EDGEBIS	en general	Garey et al, 1976
	para grafos con máximo grado 3	MacGregor, 1978
	para grafos con máximo y finito grado	MacGregor, 1978
	para grafos d-regulares	Bui et al, 1987

Tabla 4.1: Tipos de grafos que convierten a los problemas de etiquetado en NP-Completos.

para el problema del Etiquetado Lineal Mínimo en un árbol binario completo con k niveles $T_{2,k}$ adquirió la expresión descubierta por Chung: para todo $k \geq 2$,

$$MINLA(T_{2,k}) = 2^k(\frac{1}{3}k + \frac{5}{18}) + \frac{2}{9}(-1)^k - 2$$

Otra expresión recursiva fue citada por Chung para el caso de árboles ternarios completos.

Con respecto al los algortimos paralelos, Díaz et al. provaron que el problema *MINLA* para árboles en NC, puede ser resuelto en una complejidad $O(log^2 n)$ usando un *CREW PRAM* con $O(n^{3,6})$ procesadores.

El problema del *MINLA* en mallas rectangulares o cuadradas tuvo una peculiar historia. Siendo $L_{m,n}$ denotado por una malla rectangular $m \times n$ y siendo $L_n = L_{n,n}$ denotada como una malla cuadrada $n \times n$. Muradyan y Piliposyan resolvieron el problema que publicaron en un artículo escrito en Rusia para el caso general de las mallas rectangulares. Después, en 1980, Mitchison y Durbin presentaron una solución para mallas cuadradas. En 1981, Niepel y Tomasta conjeturaron incorrectamente que un etiquetado lexicográfico es la solución óptima para el $MINLA(L_m)$. En un artículo publicado en 1994 por Nakano hacía referencia a esta conjetura de nuevo. Hasta que en 2000, Fishburn et al, presentaron una nueva solución para el $MINLA(L_{m,m'})$.

El etiquetado óptimo para el $MINLA(L_{m,m'})$ en una malla rectangular $m \times m'$, tuvo una interesante solución descrita por Bezrukov en 1999. El etiquetado óptimo está mostrado en la figura 4.1. La etiquetación empieza en la parte superior izquierda de la malla y recorre las áreas $A_1, A_2, ..., A_7$, dónde A_1, A_3 son cuadrados $a \times a$ y A_5, A_7 son cuadrados $a' \times a'$; ver la parte izquierda de la figura 4.1. Los valores de a y a' deben satisfacer:

$$a, a' \in \left\{ \left\lceil m - \frac{1}{2} - \sqrt{\frac{1}{2}m^2 - \frac{1}{2}m + \frac{1}{4}} \right\rceil , \left\lfloor m + \frac{1}{2} - \sqrt{\frac{1}{2}m^2 - \frac{1}{2}m + \frac{1}{4}} \right\rfloor \right\}$$

La forma de etiquetar las áreas está mostrado en la parte derecha de la imagen 4.1. Cada cuadrado debe ser numerado secuencialmente. El cuadrado A_1 se debe rellenar empezando primero por las columnas y luego por las filas. El cuadrado A_2 debe ser rellenado eligiendo filas consecutivas, desde abajo a arriba y desde la izquierda a la derecha. El cuadrado A_3 debe ser etiquetado de forma contraria al A_1. El cuadrado A_4 debe ser etiquetado eligiendo las columnas desde abajo a arriba y desde la izquierda a la derecha. Finalmente, los cuadrados A_5, A_6 y A_7 debe ser rellenado de la misma forma. Usando este método, tenemos que:

$$MINLA(L_{m,m'}) = -\frac{2}{3}a^3 + 2ma^2 - (m^2 + m - \frac{2}{3})a + m'(m^2 + m - 1) - m$$

y

$$MINLA(L_m) = \frac{4 - \sqrt{2}}{3}m^3 + O(m^2)$$

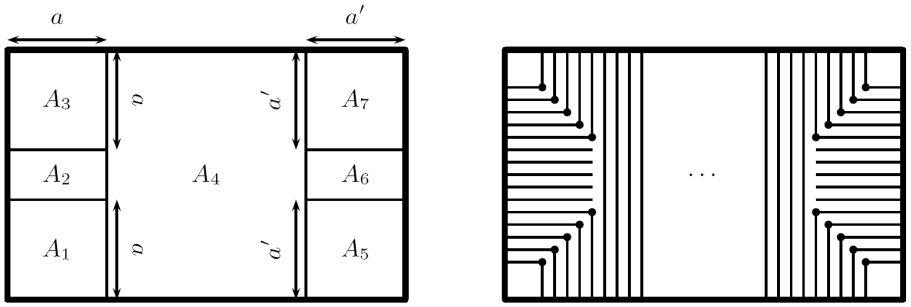


Figura 4.1: Representación esquemática del etiquetado óptimo para $MINLA(L_{m,m'})$ para una malla rectangular $m \times m'$.

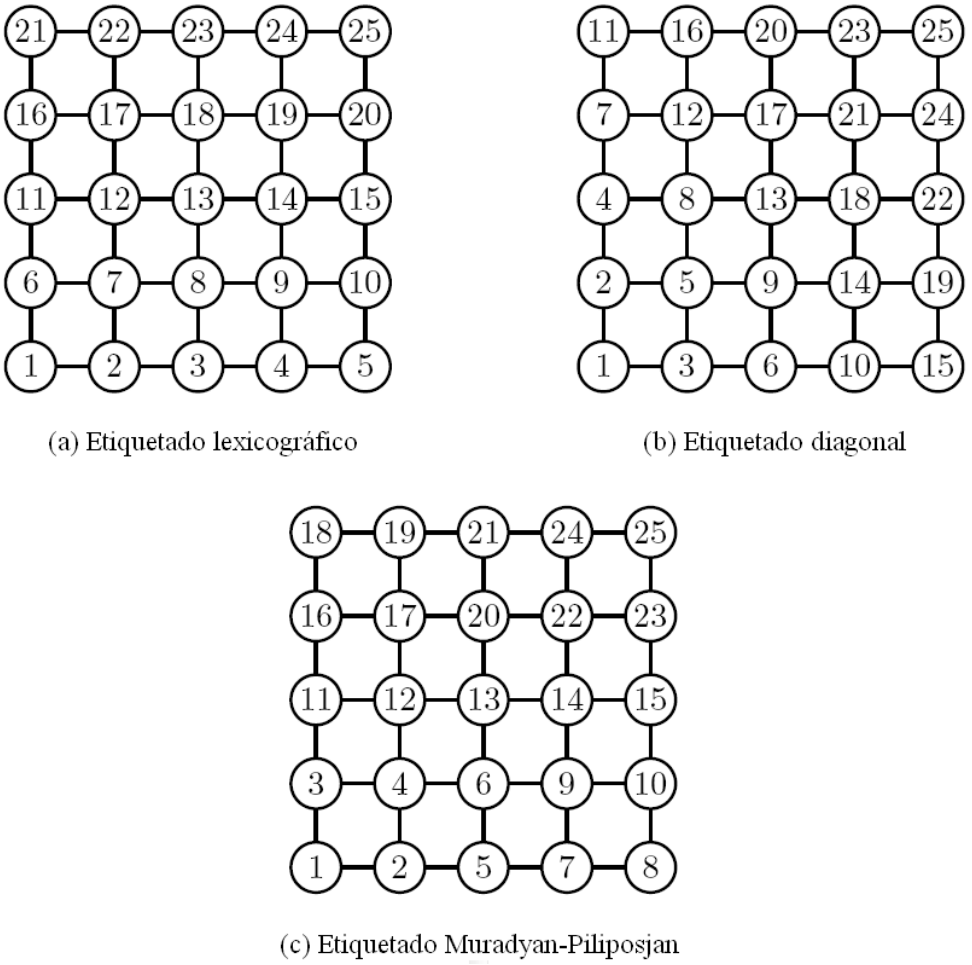
El siguiente teorema de Muradyan y Piliposyan reunen los valores óptimos para los problemas de etiquetado de mallas cuadradas:

Teorema 4 *Siendo L_m una malla cuadrada de tamaño m. Entonces,*

$$MINVS(L_m) = m, MINCW(L_m) = m + (\text{odd } m),$$

$$MINSC(L_m) = \frac{2}{3}m^3 + \frac{1}{2}m^2 - \frac{7}{6}m, MINVB(L_m) = m,$$
$$MINEB(L_m) = m + (\text{odd } m), MINLA(L_m) = \frac{1}{3}(4 - \sqrt{2})m^3 + O(m^2).$$

El etiquetado óptimo para una malla de 5×5 cuadrados está representado en la figura 4.2. El etiquetado lexicográfico es óptimo para *VERTSEP*, *BANDWIDTH*, *EDGEBIS* y *CUTWIDTH*; el etiquetado Muradyan-Piliposjan es óptimo para el *MINLA*; el etiquetado lineal es óptimo para *VERTSEP*, *VERTBIS*, *SUMCUT* y *BANDWIDTH*.



(a) Etiquetado lexicográfico

(b) Etiquetado diagonal

(c) Etiquetado Muradyan-Piliposjan

Figura 4.2: Etiquetado óptimo para una rejilla cuadrada de 5×5.

Los resultados exactos para *VERTBIS*, *SUMCUT* y *BANDWIDTH* son también conocidos por las rejillas multidimensionales citadas por Bezrukov en 1999. El problema *VERTSEP* para rejillas $n-dimensionales$ fue resuelto por Bollobás y Leader en 1991. Anotamos que nada es conocido sobre las soluciones óptimas de grafos de rejilla con agujeros, excepto por el problema de la Bisección de Aristas, para el cual Papadimitriou y Sideri proporcionaron un algoritmo con complejidad $O(n^5)$ en 1996.

Harper en 1966 fue el primero en resolver el problema del Ancho de Corte en el caso de hipercubos. Chung en 1982 presentó un algoritmo de complejidad $O(n \log^{d-2} n)$ para

el ancho de corte de árboles con n vértices y con grado máximo de d. Yannakakis en 1985 mejoró los resultados presentando un algoritmo que determina el ancho de corte de un árbol de n vértices con una complejidad de $O(n \log n)$. En el caso de un árbol de k-niveles y t-ary $T_{t,k}$, se mantiene que:

$$MINCW(T_{t,k}) = \left\lceil \frac{1}{2}(k-1)(t-1) \right\rceil, \ \forall k \geq 3$$

Los anchos de corte exactos para una rejilla de dos y tres dimensiones han sido encontrados por Rolim et al. en 1995, y además presentaron los resultados para mallas cilíndricas y toroidales. Para $m, n \geq 2$, prueban que:

$$MINCW(L_{m,n}) = \begin{cases} 2, & \text{si } m = n = 2, \\ \text{mín}\{m+1, n+1\}, & \text{en otro caso.} \end{cases}$$

Por otro lado, Thikikos et al. en 2001 presentó un algoritmo para computar el ancho de corte de grafos con grado finito en árboles con pequeña anchura en tiempo polinómico.

Con respecto a los algoritmos paralelos, Díaz et al. en 1997 provó que un etiquetado óptimo de un árbol con n vértices y grado Δ puede ser computado con una complejidad de $O(\Delta \log^2 n)$ usando un *CREW PRAM* con $O(n^{3,6})$ procesadores. Aun está por investigar la complejidad del problema paralelo del *CUTWIDTH* para árboles con infinito grado.

Para el problema *SUMCUT* o *PROFILE*, Lepin en 1986 proporcionó el primer algoritmo polinómico exacto para árboles. Estos resultados fueron mejorados por el algorimo de tiempo linear presentado por Díaz et al. en 1991. Los autores también dieron otro algoritmo para computar el óptimo etiquetado de Corte de Suma de árboles con n vértices con una complejidad de $O(\log n)$ usando *CREW PRAM* con $O(n^2 \log n)$ procesadores. Kuo y Chang en 1994 también proporcionaron un algoritmo polinómico secuencial en tiempo para el problema de los árboles.

En el caso de problema *VERTSEP*, Ellis et al. en 1979 nos dió un algoritmo lineal para computar la óptima separación entre vértices de un árbol y un algoritmo de complejidad $O(n \log n)$ para encontrar el etiquetado óptimo. Recientemente, Skodinis ha encontrado un algoritmo de tiempo lineal.

En el caso del problema de de bisección de aristas, Leighton en 1993 mostró como minimizar la longitud de bissección de productos Cartesianos de rutas con la misma longitud par. Nakano en 1994 resolvió el problema para longitudes impares. Rolim et al. en 1995 concluyó la bisección de aristas óptima para rejillas normales, cilíndricas y toroidales de dos dimensiones y para rejillas normales y toroidales de tres dimensiones. El siguiente teorema muestra los resultados para $L_{m,n}$:

Teorema 5 *Siendo $L_{m,n}$ una rejilla rectangular. Para $2 \leq m \leq n$,*

$$MINEB(L_{m,n}) = m + (odd\ n)$$

y, para $m \geq 2, n \geq 3$,

$$MINEB(L_{m,n}) = \begin{cases} \text{mín}\{2m, n\}, & \text{si } m \text{ y } n \text{ son pares,} \\ \text{mín}\{2m, n+2\}, & \text{si } m \text{ es impar y } n \text{ es par,} \\ \text{mín}\{2m+1, n\}, & \text{si } m \text{ es par y } n \text{ es impar,} \\ \text{mín}\{2m+1, n+2\}, & \text{si } m \text{ y } n \text{ son impares.} \end{cases}$$

La bisección de un hipercubo parece haber sido resulto por muchas personas simultaneamente. Manabe et al. en 1984 nos ofrecen una fórmula para la bisección de aristas de un grafo con ciclos de cubos conectados. Para el caso de los árboles, MacGregor en 1978 nos proporcionó un algoritmo con complejidad $O(n^3)$. Goldberg y Miller en 1988 presentaron un algoritmo paralelo con complejidad $O(\log^2 n \log\log n)$ en un *CRCW PRAM* con $O(n^2)$ procesadores para la bisección de árboles, el cual estaba basado en el algoritmo de MacGregor. Soumyanath y Deogun en 1990 presentaron un algoritmo de complejidad $O(n^2)$ para computar una óptima bisección de un k-árbol parcial. Un k-árbol parcial es un grafo con finita anchura.

Muradyan y Piliposyan en 1980 resolvieron los problemas *MINLA* y *CUTWIDTH* con grafos completos p-partitos. Los grafos completos p-partitos $K(N_1, ..., N_p)$ son grafos en los cuales su conjunto de vértices pueden ser particionados dentro de P, es decir, dos vértices pueden ser adyacentes si y solo si pertenecen a diferentes particiones. Si el número de vértices en dos diferentes particiones difiere en más de uno, a un p-partito se le llama balanceado. Se podría denotar como que el $MINLA(K(N_1, N_2, ..., Np))$ tiene una solución anidada.

Como dijimos anteriormente, el problema del ancho de banda es un problema NP-completo cuando se basa en árboles. Sin embargo, en el caso de un árbol de k-niveles y t-ary $T_{t,k}$ mantiene que:

$$MINBW(T_{t_k}) = \left\lceil \frac{t(t^{k-1}-1)}{2(k-1)(t-1)} \right\rceil$$

Heckmann et al. en 1998 presentaron una óptima forma de resolver el ancho de banda de un árbol binario completo. Existe un algoritmo de complejidad $O(nlogn)$ para determinar el ancho de banda de orugas con el tamaño del pelo como máximo de dos.

Otra clase de grafos en los cuales su ancho de banda puede ser computado eficientemente son: grafos de intervalo, mariposas y grafos en cadena. Los grafos de intervalo son intersecciones de grafos de un conjunto de intervalos superpuestos en una línea real, y los grafos

en cadena son grafos bipartitos $G = (X, Y, E)$ donde hay una ordenación $x_1, x_2, ..., x_{|X|}$ de X tal que $\Gamma(x_1) \subseteq \Gamma(x_2) \subseteq ... \subseteq \Gamma(x_{|X|})$. El primer artículo que contenía un algoritmo polinómico para calcular el ancho de banda de grafos de intervalo fue el de Muradyan en 1986. El segundo artículo fue el de Kratsch en 1987. El tercer algoritmo fue propuesto por Kleitman y Vohra en 1990 el cual tenía una complejidad de $O(n\text{BW}(G))$. Se descubrieron defectos en las pruebas de los algoritmos de Kratsch en dos artículos independientes, el de Mahesh et al. en 1991 y en el de Sprague en 1994. Mahesh et al. crearon en 1991 un algoritmo de complejidad $O(n^2)$ que ampliaba los resultados obtenidos en el artículo de Keitman y Vohra de 1990. El problema *SUMCUT* es otro tipo de problema de etiquetado con un algoritmo de tiempo polinómico que fue propuesto por Lin y Yuan en 1994.

La tabla 4.2 nos muestra un resumen de todas las clases de grafos con sus algoritmos de tiempo polinómico óptimos para los problemas de etiquetado.

Aparte de rejillas, otro tipo de familias de grafos pueden ser descritos aplicando operaciones de composición mediante rutas (P_n), ciclos (C_n), árboles (T_n), grafos completos (K_n) o grafos completos bipartitos ($K_{n,m}$). Podemos encontrar todas estas definiciones en el artículo de Lai y Williams de 1999.

- El $r-th$ poder de un grafo G es un grafo G^r, con el mismo conjunto de vértices de G y un arista uv si $d(u, v) \leq r$ en G.
- La *suma* de grafos pares disjuntos $k \geq 2$, $G_1, ..., G_k$ denota que $G_1 + ... + G_k$, es el grafo G con el conjunto de vértices $\cup_{1 \leq i \leq k} V(G_i)$ y un arista uv si para algún $i \neq j$ $u \in V(G_i)$ y $v \in V(G_j)$ o si para algún i $u, v \in V(G_i)$ y $uv \in E(G_i)$.
- El *producto cartesiano* de dos grafos G y H, denotado como $G \times H$, es el grafo que tiene un conjunto de vértices $V(G) \times V(H)$ donde (u_1, v_1) es adyacente a (u_w, v_2) si bien $u_1u_2 \in E(G)$ y $v_1 = v_2$ o $v_1v_2 \in E(H)$ y $u_1 = u_2$.
- La *composión* de dos grafos G y H, denotado como $G[H]$, es el grafo con el conjunto de vértices $V(G) \times V(H)$ donde (u_1, v_1) es adyacente a (u_2, v_2) si bien $u_1u_2 \in E(G)$ o $v_1v_2 \in E(H)$ y $u_1 = u_2$.
- El *producto fuerte* de dos grafos G y H, denotado como $G \odot H$, es el grafo con el conjunto de vértices $V(G) \times V(H)$ donde (u_1, v_1) es adyacente a (u_2, v_2) si uno de los siguientes se mantiene: (a) $u_1u_2 \in E(G)$ y $v_1v_2 \in E(H)$, (b) $u_1 = u_2$ y $v_1v_2 \in E(H)$, o (c) $v_1 = v_2$ y $u_1u_2 \in E(G)$.
- La *corona* de dos grafos G y H es denotada como $G \wedge H$ y contiene una copia de G y una copia de H para cada vértice de G. Cada vértice de G está conectado a cada vértice correspondiente en la copia de H.

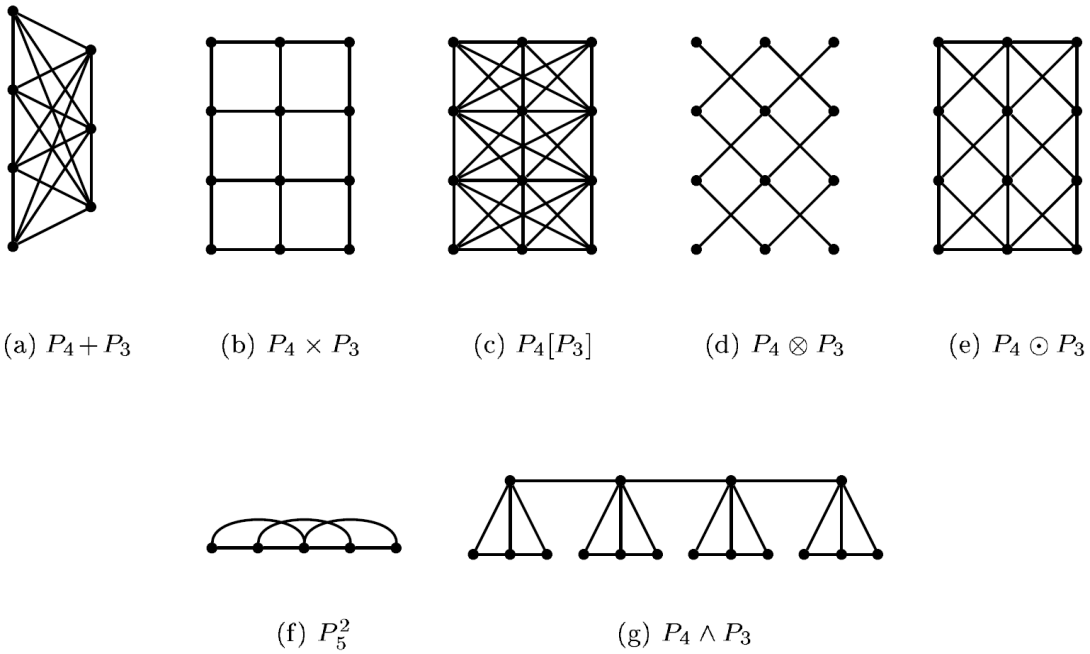
  En la figura 4.3 se muestra estas definiciones y la tabla 4.3 hace un resumen de los resultados conocidos.

Problema	Tipo de grafo	Complejidad	Autor
BANDWITH	orugas con tamaño de pelo ≤ 2	$O(n \log n)$	Assman et al, 1981
	hipercubos	$O(n \log n)$	Harper, 1966
	mariposas	$O(n \log n)$	Lai, 1997
	grafos de intervalo	$O(n\Delta^2 \log \Delta)$	Muradyan, 1986
	grafos de intervalo	$O(n \log n)$	Mahesh et al, 1991
	grafos de intervalo	$O(n \log n)$	Sprague, 1994
	grafos en cadena	$O(n^2 \log n)$	Kloks et al, 1998
	árboles completos de k-niveles y t-ary	$O(n)$	Heckmann et al, 1998
	rejillas cuadradas	$O(n)$	Mai y Luo, 1984
			Díaz et al, 2000
MINLA	árboles	$O(n^3)$	Goldberg y Kilpker, 1976
	árboles con raices	$O(n \log n)$	Adolphson y Hu, 1973
	árboles	$O(n^{2,2})$	Shiloach, 1979
	árboles	$O(n^{\log 3/\log 2})$	Chung, 1988
	rejillas rectangulares	$O(n)$	Muradyan y Piliposyan, 1980
	rejillas cuadradas	$O(n)$	Mitchison y Durbin, 1986
	cilindros de dos dimensiones	$O(n)$	Muradyan, 1982
	hipercubos	$O(n)$	Harper, 1964
	grafos Bruijn de orden 4	$O(n)$	Harper, 1970
	cliques d-dimensionales y c-ary	$O(n)$	Lindsey, 1964
	grafos completos p-partitos	$O(n + p\log(p))$	Muradyan y Piliposyan, 1980
CUTWIDTH	árboles	$O(n \log^{\Delta-2} n)$	Chung et al, 1982
	árboles	$O(n \log n)$	Yannakakis, 1985
	hipercubos	$O(n)$	Harper, 1964
	cliques d-dimensionales y c-ary	$O(n)$	Nakano, 1994
	max. grado $\leq \Delta$ y anchura de árbol $\leq k$	$O(n^{\Delta k^2})$	Thilikos et al, 2001
	mallas de 2 y 3 dimensiones	$O(n^2)$	Rolim et al, 1995
	mallas toroidales y cilindricas de dos dimensiones	$O(n^2)$	Rolim et al, 1995
	mallas toroidales de tres dimensiones	$O(n^2)$	Rolim et al, 1995
	grafos completos p-partitos	$O(n + p\log(p))$	Muradyan y Piliposyan, 1980
VERTSEP	árboles	$O(n \log n)$	Ellis et al, 1979
	árboles	$O(n)$	Sokodinis, 2000
	cografos	$O(n)$	Bodlaender y Mohring, 1993
	grafos permutados	$O(n^2)$	Bodlaender et al, 1995
	rejillas n-dimensionales	$O(n^2)$	Bollobás y Leader, 1991
SUMCUT	árboles	$O(n^{2,3})$	Lepin, 1986
	árboles	$O(n)$	Díaz et al, 1991
	árboles	$O(n^{1,722})$	Kuo y Chang, 1994
	rejillas cuadradas	$O(n)$	Díaz et al, 2000
	grafos de intervalo		Lin y Yuan, 1994
EDGEBIS	árboles	$O(n^3)$	MacGregor, 1978
	hipercubos	$O(n)$	Nakano, 1994
	arrays d-dimensionales y c-ary	$O(n)$	Nakano, 1994
	cliques d-dimensionales y c-ary	$O(n)$	Nakano, 1994
	mallas de dos y tres dimensiones	$O(n^2)$	Rolim et al, 1995
	mallas toroidales y cilíndricas de dos dimensiones	$O(n^2)$	Rolim et al, 1995
	grafos de rejilla	$O(n^5)$	Papadimitriou y Sideri, 1996
	anchura de árboles $\leq k$	$O(n^2)$	Soumyanath y Deogun, 1990
	grafos con ciclos de cubos conectados	$O(n)$	Manabe et al, 1984

Tabla 4.2: Resumen de grafos resultos óptimamente con algoritmos de tiempo polinómico (n indica el número de vértices del grafo, m es el número de aristas y Δ es el grado máximo).

4.3 Resultados de parametrización fija

En la definición de parametrización encontramos que el problema no se basa el lo difícil que un problema pueda ser, sino si el problema es difícil o fácil de computar. Al estu-



(a) $P_4 + P_3$ (b) $P_4 \times P_3$ (c) $P_4[P_3]$ (d) $P_4 \otimes P_3$ (e) $P_4 \odot P_3$

(f) P_5^2 (g) $P_4 \wedge P_3$

Figura 4.3: Ejemplos de composición de grafos: *suma* (a), *producto cartesiano* (b), *composición* (c), *producto tensor* (d), *producto fuerte* (e), *poder* (f) y *corona* (e).

diar la dureza estructural de un problema difícil, nos centramos en dividir la cuestión en dos maneras: la parte difícil (la no parametrizable) y la parte *fácil* (la parametrizable), dónde imponemos algunas restricciones. Para algunos problemas, es sabido que la parametrización de las entradas no rompe la barrera de los problemas NP-Completos. Un ejemplo clásico es el *problema del colorizado* parametrizado por un número k de colores que pueden ser usados. El problema se convierte en un problema NP-completo para $k \geq 3$. Por otro lado, los problemas de grafos se vuelven polinómicamente más fáciles de resolver, para cada k, cuando los parametrizamos según el tamaño k del conjunto máximo independiente o la cobertura de vértices mínima.

Incluso en los casos de parametrización de un problema NP-Completo se acercan a ser algoritmos de tiempo polinómico. Hay diferentes tipos de límites superiores en la ejecución de los mejores algoritmos conocidos. Por ejemplo, si el problema tiene k como un parámtro, el tiempo de ejecución podría ser $O(n^{f(k)})$ o podría ser $O(f(k)n^{\alpha})$ o $O(f(k) + n^{\alpha})$, donde $f(k)$ es una función de k y $\alpha \in N$. Se dice que un problema de parametrización es de *Parametrización Fija Manejable* si existe un algoritmo de complejidad $O(f(k)n^{\alpha})$ que resuelve el problema. La clase FPT es la clase de todos los problemas de parámetros fijos manejables. Downey y Fellows definieron una herencia compleja de parametrización, la W-herencia. Similar a la teoría desarrollada en la Complejidad Estructural, la W-herencia consta de clases para problemas de parametrización, con diferentes niveles de estructuras de parametrización, entre FPT y $W[P] : FPT \subseteq W[1] \subseteq W[2] \subseteq ... \subseteq W[P]$, a lo largo de una adecuada integridad.

Problema	Tipo de grafo	Autor
BANDWIDTH	$G_1 + ... + G_k$	Lai et al, 1994
	$P_m \times ...^d \times P_n$	Chvátalová, 1975
	$C_n \times C_m$	Lai y Williams, 1995
	$K_n[G], P_n[G], C_n[G]$	Hendrich y Stiebitz, 1992
	$K_{1,n}[G], K_n[G]$para$G \in \{P_m, T_m, C_m\}$	
	ó $G = C_{n_1} \times ... \times C_{n_k}$ con $n_i \leq 5$	
	ó $G = P_{n_1} \times ... \times P_{n_k}$ con $n_i > 1$	Liu y Williams, 1995
	$P_n^r[G], C_n^r[G]$	Chinn et al, 1995
	$P_n \times P_m[G], P_n \times C_m[G]$ con $2n \neq m$	
	y $C_n \times C_m[G]$ $6 \leq 2n \leq 2s$	Zhou y Yuan, 1998
	$K_n \odot P_m$	Hendrich y Stiebitz, 1992
	$P_n \odot P_m, C_n \odot P_m, P_n \odot C_m, C_n \odot C_m$	Lai y Williams, 1995
	$P_n \otimes P_m, C_n \otimes P_m, C_n \otimes C_m$	Lai y Williams, 1997
	$P_k \otimes K_{nm}$	Williams, 1994
	$P_k \otimes K_n, C_k \otimes K_n$	Williams, 1996
	$K_n \wedge K_m, C_n \wedge K_1, P_n \wedge K_1, C_n \wedge (K_1 \cup ... \cup K_1)$	Chinn et al, 1992
MINLA	$G_1 + ... + G_k$ para todo G_i que sea suma determinista	Lai y Williams, 1994
	$P_n \times C_m$	Muradyan, 1982
	$P_n[P_m], P_n[C_m]$	Liu, 1992
	$P_n[P_m], K_n[C_m]$	Liu y Williams, 1995
	$P_k \otimes L_{nm}$	Williams, 1994
	$P_n \wedge P_m, K_n \wedge K_1$	Williams, 1993
CUTWIDTH	$C_m^r, P_n \times P_m, P_n \times C_m, C_n \times C_m, K_n \times P_m$	
	y $K_n \times C_m, C_n^s \times C_m^r, K_n \times K_m, P_n \odot C_m$	Liu y Yuan, 1995
SUMCUT	$G_1 + G_2$	Lin y Yuan, 1994
	$P_n \times K_m, C_n \times K_m, C_n \times C_m$	Mai, 1996
	$K_n[G], P_n[G], C_n[G]$	Lai, 1997
	$P_k \times K_{nm}$	Lai, 2001
	$P_n \wedge G, C_n \wedge G, K_n \wedge G, K_{nm} \wedge G$	Lai, 1997
	$G \wedge H$ donde H es un 1-oruga, K_n o C_n	Chang y Lai, 2001

Tabla 4.3: Revisión de las familias de grafos optimizados resueltos en tiempo polinómico o con una fórmula. Los nombres de los grafos y los operadores están descritos en el texto.

Hay algunos resultados complejos de parametrización para problemas de etiquetado. Los resultados de estos problemas se parametrizan por el manor valor obtenido en el etiquetado. Para el caso del *BANDWIDTH*, consideramos la siguiente parametrización:

BANDWIDTH(k): Dado un grafo G, determinar si MINBD(G) $\leq k$.

Usaremos la misma notación y la misma parametrización para los problemas de etiquetado restantes.

El resultado más fácil probado es el de si es posible decidir el *BANDWIDTH(2)* y el *CUTWIDTH(2)* en una línea temporal. Con respecto al problema del *BANDWIDTH*, Saxe en 1980 presentó un algoritmo de complejidad $O(n^{k+1})$ para decidir el *BANDWIDTH(k)* para una constante k dada. Gurari y Sudborough en 1984 mejoraron estos resultados gracias a su algoritmo de complejidad $O(n^k)$. Este resultado fue el mejor obtenido, ya que se probó que, para cualquier k, el problema del *BANDWIDTH0* es W[k]-duro. Existen estudios recientes sobre la simplificación del algoritmo para el problema del *BANDWIDTH(2)*, para grafos biconectados y para grafos generales.

En el caso del problema del *CUTWIDTH(k)*, Gurari y Sudborough en 1984 presentaron un algoritmo de complejidad $O(n^k)$ para decidir el grafo de entrada con n vértices y cualquier constante k. Los resultados fueron mejorados después por Makedon y Sudborough en 1989 con un algoritmo de complejidad $O(n^{k-1})$. Más tarde, Fellows y Langston en 1988 obtuvieron un algoritmo de complejidad $O(n^2)$. El resultado para el problema del *CUTWIDTH(k)* ha sido recientemente mejorado por Thilikos et al. en 2000, dónde se obtuvo un algoritmo de complejidad lineal. Señalamos que la noción de la anchura de un árbol es parecida a la noción de anchura de un camino, pero para la descomposición de un árbol y para la descomposición de un camino.

Por otra parte, Fellows y Langston en 1988 probaron que los problemas del *VERTSEP* y *MODCUT* podían ser manejados mediante parametrización fija. En particular, Bodlaender en 1996 probó que el problema del *VERTSEP(k)* podía ser decidido en tiempo lineal.

La tabla 4.4 hace una descripción de todo lo visto anteriormente.

Problema	**Complejidad**	**Autor**
BANDWIDTH(2)	$O(n)$	Gare et al, 1978
BANDWIDTH(k)	$O(n^{k+1})$	Saxe, 1980
BANDWIDTH(k)	$O(n^k)$	Gurari y Sudborough, 1984
BANDWIDTH(k)	$W[k]$	Bodlaender et al, 1994
CUTWIDTH(2)	$O(n)$	Garey et al, 1978
CUTWIDTH(k)	$O(n^k)$	Gurari y Sudborough, 1984
CUTWIDTH(k)	$O(n^{k-1})$	Makedon y Sudborough, 1989
CUTWIDTH(k)	$O(n^2)$	Fellows y Langston, 1992
CUTWIDTH(k)	$O(n)$	Thilikos et al, 2000
MODCUT(k)	$O(n^2)$	Fellows y Langston, 1992
VERTSEP(k)	$O(n^2)$	Fellows y Langston, 1988
VERTSEP(k)	$O(n)$	Dodlaender, 1996

Tabla 4.4: Resultados de la parametrización fija para problemas de etiquetado (n indica el tamaño del grafo y k el parámetro).

4.4 Algoritmos de aproximación

Un de los enfoques a tratar con problemas intratables es diseñar un algoritmo de aproximación que en tiempo polinómico nos de una solución factible cercana a la óptima. En esta sección mostraremos algunos de los resultados obtenidos sobre esta idea. En los artículos de Garey y Johnson de 1979, Ausiello et al. en 1999 y en el de Vazirani 2001 podemos encontrar los textos completos sobre estas teorías.

Recalcamos que, dado un problema de minimización Π, un algoritmo de $r(n)$-aproximaciones es un algoritmo que, para una entrada x de tamaño n, encuentra una solución a Π la cual su coste es al menos $r(n)$ veces el coste de la solución óptima del problema. Cuando un problema Π tiene un algoritmo $r(n)$-aproximaciones, se dice que es un $r(n)$-aproximable.

Cuando existe un algoritmo para Π tal que, para todo $\epsilon < 1$, A_ϵ devuelve una solución factible σ tal que el radio entre el valor obtenido y el valor optimo es menor de $1 + \sigma$ y se ejecuta en tiempo polinómico con respecto a $|x|$, se dice que A_σ es un *esquema de aproximación en tiempo polinómico*. Además, cuando A_ϵ se ejecuta en tiempo polinómico con respecto a $|x|$ y $1/\epsilon$, se dice que A_ϵ es un esquema de aproximación en tiempo polinómico completo. Un problema de optimización combinatoria pertenece a la clase **APX** si es σ si es ϵ-aproximable para una constante $\epsilon > 1$, a la clase **PTAS** si admite un esquema de aproximación y a la clase **FPTAS** si admite un esquema de aproximación completo. Además **FPTAS** $\subseteq$ **PTAS** $\subseteq$ **APX**, dónde las inclusiones son estrictas si y solo si **P** $\neq$ **NP**. Hay otras clases similares de aproximación en el artículo de Díaz et al. de 1997.

Para los problemas de etiquetado de grafos, el conjunto de instancias I corresponde al conjunto de todos los grafos no dirigidos, una instancia $x \in I$ corresponde a un particular grafo no dirigido G, el conjunto de soluciones factibles $S(G)$ corresponde a $\phi(G)$, y la función objetiva es un coste de etiquetado $f \in \{LA, BW, SC, VS, CW, MC, EB, VB\}$.

Otros estudios recientes con resultados optimistas ponen en énfasis la mejora hacia una aproximación a los límites, tales que mejoran además la complejidad. Una *matriz de difusión* en un grafo es una asignación de longitudes a las aristas o a los vértices de manera que se obtienen subgrafos de manera no trivial en relación con el espacio métrico. El volumen de difusión metrica, definida como la suma de las longitudes de todos los aristas o vértices, proporciona un límite inferior al resolver problemas que usan técnicas de divide y vencerás. El inconveniente de los algoritmos basados en difusión métrica es que necesitan resolver un programa linear con un número exponencial de limitaciones, los cuales puede hacer que ésto sea impracticable. De hecho, el tiempo de ejecución es denominado por la complejidad de encontrar un difusión métrica que pueda ser computada por algoritmos de programación lineal generales, como el método *Ellipsoid*. En general, la mayoría de ellos toman un tiempo de $\tilde{O}(m^2 n)$, donde $\tilde{\text{O}}$ ignora los factores poli-logaritmicos. Las *métricas de flujo* son un nuevo enfoque a la aproximación de los problemas de etiquetado, citados en el artículo de Bornstein y Vempala en 2002. Incluso si las métricas de flujo no mejoran los resultados previos, éstas representan un entorno de trabajo más simple donde el número exponencial de limitaciones se reduce a un número polinómico.

Para el problema del *BANDWIDTH*, algunos tipos particulares de grafos tienen algoritmos de aproximación. En el caso de los grafos γ-densos, existen un tiempo polinómico de 3-aproximaciones, citadas en el artículo de Karpinsky et al. en 1997. Recalcamos que un grafo con n vértices es γ-denso si su mínimo grado es al menos γn. Hay un algoritmo de 2-aproximaciones para el ancho de banda de grafos libres triples asteroidales, para grafos de permutación y para grafos trapezoidales. También hay algoritmos de $O(\log n)$-aproximaciones para orugas y para muchos tipos de árboles, denotados como árboles-GHB, los que están caracterizados como árboles tales que para cualquier nodo v, la diferencia de profundidad de cualquier subárbol no vacio en v es finito por una constante,

citado en Haralambides y Makedon en 1997. Para el caso de árboles generales y grafos de cuerdas, Gupta en 2001 presentó un algoritmo aleatorio de $O(\log^{2,5} n)$-aproximaciones. Para los grafos generales, el ancho de banda tiene varios algoritmos de aproximación polilogarítmica ejecutándose en tiempo aleatorio polinómico (en el artículo de Blum et al. de 2000), usando relajación semidefinitiva (en el artículo de Feige en 2000), usando difusión métrica y volumen (en el artículo de Dunagan y Vempala en 2001), usando relajación semidefinitiva y modelos Euclideos. De una forma negativa, Blanche et al. en 1998 nos muestran que es NP-Completo encontrar una $\frac{4}{3}$-aproximaciones para árboles. Como consecuencia, el *BANDWIDTH* no pertenece a la **PTAS**. De hecho, Unger en 1998 prueba que, para cualquier constante k, es NP-Completo encontrar k-aproximaciones en las orugas. De esta forma, *BANDWITH* tampoco pertenece a **APX**. La aproximición del *BANDWIDTH* entre un factor constante y uno polinómico es un campo abierto en la actualidad.

Recordamos que los grafos con n vértices y m aristas son densos si $m = \Theta(n^2)$. Existen esquemas de aproximación en tiempo polinómico total para grafos densos para *MINLA* y *CUTWIDTH*, descubiertos por Arora en 1996 y 1999, y para el *CUTWIDTH* descubierto por Frieze y Kannan en 1996. El Lema de Regularidad de Szemmerédi tiene la técnica base para probar estos resultados.

Para los problemas *MINLA*, *CUTWIDTH* y *SUMCUT* se han diseñado algunos algoritmos de aproximación. El primer, no trivial, algoritmo de aproximación para el *MINLA* y el *CUTWIDTH* en grafos normales fueron unos algoritmos de complejidad $O(\log n)$ para encontrar el balanceamiento de particiones en un grafo, propuesto por Leighton y Rao en 1999. Usando las ideas de trabajos anteriores, Hansen en 1989 proporcionó un algoritmo de $O(\log^2 n)$-aproximaciones para el *MINLA*. Usando las técnicas de las métricas de difusión, Even et al. en 2000 ofreció un algoritmo de $O(\log n \log\log n)$-aproximaciones para el *MINLA* y el *SUMCUT*. Avanzando en el tiempo, el mejor tiempo conseguido en un algoritmo polinómico de aproximación para el *MINLA* y el *SUMCUT* tiene $O(\log n)$-aproximaciones para grafos generales y $O(\log\log n)$-aproximaciones para grafos planos. Ambos resultados aparecen en el artículo de Rao y Richa de 1998, los cuales usan técnicas de métricas de difusión.

En el caso del problema del *VERTSEP*, Bodlaender et al. en 1995 presentaron un algoritmo en tiempo polinómico de $O(\log^2 n)$-aproximaciones para grafos generales, y mostraron como usar los resultados de Sevmour y Tomas de 1994 para obtener un algoritmo de $O(\log n)$-aproximaciones para grafos planos.

El primer algorimo de aproximación para el problema del *EDGEBIS* en grafos generales con un ratio de aproximación lineal fue nombrado por Feige et al. en 2002. Estos resultados fueron mejorados después por Feige y Krauthgamer en 2002, donde se obtenía un ratio de aproximación de $O(\log^2 n)$.

La tabla 4.5 presenta un resumen de lo anteriormente expuesto.

Problema	Aproximidad	Complejidad	Autor
BANDWITH	3-aproximaciones para un grafo δ-denso	$n^{O(1/\delta)}$	Karpinsky et al, 1997
	2-aproximaciones para un grafo AT-libre	$O(n^3)$	Kloks et al, 1999
	$O(\log n)$-aproximaciones para orugas	$O(n^2)$	Haralambides et al, 1991
	$O(\log n)$-aproximaciones para árboles-GHB	$O(n^2)$	Haralambides y Makedon, 1997
	$O(\log^{4,5} n)$-aproximaciones aleatorias	$O(m(\log n)^4 \log\log n)$	Feige, 2000
	$O(\log^3 n\sqrt{\log\log n})$-aproximaciones aleatorias	pol(n)	Dunagan y Vempala, 2001
	$O(\sqrt{n/\mathrm{BW}(G)}\log n)$-aproximaciones aleatorias	pol(n)	Blum et al, 2000
	$O(\sqrt{\log^{2,5} n}$-aproximaciones aleatorias para árboles y grafos de cuerdas	pol(n)	Gupta, 2001
	sin **PTAS**	-	Blanche et al, 1998
	sin **PTAS** para árboles	-	Blanche et al, 1998
	sin **APX**	-	Unger, 1998
VERTSEP	$O(\log^2 n)$-aproximaciones	pol(n)	Bodlaender et al, 1995
	$O(\log n)$-aproximaciones para grafos planos	pol(N)	Bodlaender et al, 1995
MINLA	**PTAS** para un grafo denso	$n^{O(1/\epsilon^2)}$	Arora et al, 1996
	$O(\log^2 n)$-aproximaciones	pol(n)	Hansen, 1989
			Leighton y Rao, 1999
	$O(\log n \log\log n)$-aproximaciones	pol(n)	Even et al, 2000
	$O(\log n)$-aproximaciones	pol(n)	Rao y Richa, 1998
	$O(\log\log n)$-aproximaciones para grafos planos	pol(n)	Rao y Richa, 1998
CUTWIDTH	**PTAS** para grafos densos	$n^{O(1/\epsilon^2)}$	Arora et al, 1996
	$O(\log^2 n)$-aproximaciones	pol(n)	Leighton y Rao, 1999
SUMCUT	$O(\log n \log\log n)$-aproximaciones	pol(n)	Even et al, 2000
	$O(\log n)$-aproximaciones	pol(n)	Rao y Richa, 1998
	$O(\log\log n)$-aproximaciones para grafos planos	pol(n)	Rao y Richa, 1998
EDGEBIS	**PTAS** para grafos densos	$\tilde{O}(1/\epsilon^2)n + 2^{\tilde{O}(1/\epsilon^2)}$	Frieze y Kannan, 1996
	$O(\log^2 n)$-aproximaciones	pol(n)	Feige y Krauthgamer, 2002
	$O(\log n)$-aproximaciones	pol(n)	Feige y Krauthgamer, 2002

Tabla 4.5: Revisión de los resultados de aproximidad para problemas de etiquetado (n es el número de vértices del grafo de entrada, m es el número de aristas y ϵ es el parámetro del esquema de aproximación).

5. METODOLOGÍA

5.1 Recocido Simulado

El Recocido Simulado (*Simulated Annealing*) es una metaheurística trayectoria que se basa en la analogía que puede existir entre un proceso de optimización combinatoria y un proceso termodinámico, conocido como *recocido*. Este proceso consiste en elevar la temperatura de un sólido cristalino con defectos hasta una temperatura determinada, que por lo general suele ser alta. Posteriormente, se permite que el material se enfrie muy lentamente en un *baño térmico*. El proceso de enfriamiento viene descrito por una función de la temperatura conocida como *cola de enfriamiento*, que generalmente suele ser continua y suave. Con este proceso se pretende que el sólido alcance una configuración de red cristalina lo más regular posible, eliminando durante este proceso los posibles defectos que tuviese originalmente. La nueva estructura cristalina se caracteriza por tener un estado de energía de red mínimo.

Desde un punto de vista algorítmico, el principio de operación en el que se basa el SA se puede enunciar en los siguientes términos:

SA es un algoritmo de búsqueda local capaz de escapar de los óptimos locales permitiendo que bajo ciertas circunstancias se admitan movimientos que empeoren la mejor solución encontrada hasta el momento en el proceso de búsqueda.

Así pues, el recocido simulado es un procedimiento de búsqueda local que introduce una fase de aleatorización en la aceptación de movimientos, de tal forma que si el movimiento es de mejora se acepta. Por el contrario, si el movimiento conduce a una solución peor, se acepta con la probabilidad dada. La aleatorización en la aceptación de movimientos viene controlada por el parámetro T, que se corresponde con la temperatura en el proceso termodinámico. La temperatura permita que en los primeros instantes de la búsqueda la mayoría de los movimientos se acepten, aunque éstos empeoren la solución. Posteriormente, la temperatura se va reduciendo, lo cual implica que cada vez se hace más restrictivo el proceso de aceptación de estados peor calidad.

Tanto la elección de la temperatura inicial como de la función que se utiliza para describir la cola de enfriamiento, son factores críticos en el diseño de SA. En la tabla 5.1 podemos ver algunos métodos de enfriamiento.

Siguiendo con la analogía termodinámica, SA empieza la búsqueda de estado más estable con una temperatura elevada; por lo tanto, la probabilidad de aceptar estados «*peores*» es alta. En esta situación inicial no es demasiado «*peligroso*» aceptar soluciones

Mecanismo	Función
En sucesivas temperaturas descenciendes predefinidas	$t_i = s_i, s = \{t_1, ..., t_n\}$
Enfriamiento constante	$t_i = k$
Descenso geométrico	$t_{i+1} = \alpha \cdot t_i, \alpha \in [0{,}8, 0{,}99]$
Criterio de Boltzmann	$t_i = t_0/(1 + log(i))$
Esquema de Cauchy	$t_i = t_0/(1 + i)$
Lundi y Mees	$t_{i+1} = t_i/(1 + \beta \cdot t_i)$, con β pequeña

Tabla 5.1: Ejemplos de mecanísmos de enfriamiento en SA.

peores, ya que se supone que la búsqueda está demasiado lejos del óptimo global. Posteriormente, esta temperatura se va reduciendo (cola de enfriamiento), con lo que la probabilidad de aceptación de estados peores va disminuyendo. Se supone que según se va disminuyendo la temperatura se está más cerca de óptimo global y que las partículas del sólido van ocupando sus posiciones definitivas. Este proceso termina cuando se encuentra una solución satisfactoria. En el algoritmo 5.2 se presenta el pseudo-código de alto nivel para la metahurística SA.

5.2 GRASP

El nombre de esta metaheurística viene de su acrónimo en inglés *Greedy Randomized Adaptive Search Procedure* (GRASP), que en castellano se podría traducir como *procedimientos de búsqueda miope (constructiva, voraz o ávida), aleatorizados y adaptativos* [2]. Cada uno de los términos incluidos en el nombre se corresponde con la característica distintiva de la metaheurística. GRASP se basa en el siguiente principio de operación:

GRASP es un procedimiento multi-arranque en el que cada arranque se corresponde con una iteración. Cada iteración tiene dos fases bien diferenciadas: la fase de construcción, que se encarga de obtener una solución factible de alta calidad; la fase de mejora, que se basa en la optimización (local) de la solución obtenida en la primera fase.

Originalmente GRASP fue desarrollado por T. Feo y M. Resende como un algoritmo para resolver problemas de *cubrimiento de conjuntos*. No fue hasta el trabajo que se publicó en el año 1995 cuando adquiere una terminología y forma definitiva como metaheurística de propósito general.

Los orígenes algoritmicos de GRASP provienen de la *metaheurística semi-constructiva (semi-constructive heuristic)*. Esta técnica también se caracteriza por ser un método multi-arranque basado en una construcción miope aleatorizada. La diferencia fundamental con respecto a GRASP es que la primera técnica no utilizaba un procedimiento de mejora (búsqueda local).

GRASP es un procedimiento multi-arranque, donde cada iteración está compuesta de

```
{x: TipoSolucion} = SA(T: real; M_max: integer; f: TipoSolucionObjetivo)
/*M_max son las iteraciones por temperatura */
var
  x,y: TipoSolucion;
  V: list of TipoSolucion; // Estructura de vecindad;
  t_k: (cola de enfriamiento);
  k: integer; // Indice de cambio de temperatura

  begin
    {x} := SolucionInicial(); // Inicializar solucion
    k = 0;
    {t_k} = ColaEnfriamiento; // Inicializar el enfriamiento
    k: integer; // Indice de cambio de temperatura
    repeat
      m = 0;
      repeat
        {y} = SeleccionarVecino(V(x));
        ΔE = f(y) − f(x);
        if ΔE ≥ 0 then
          x = y;
        else
          ρ = rand(1);
          if ρ ≤ e^(δE/t_k) then
            x = y;
          end if
        end if
        m = m + 1;
      until m = M_max
      {t_k} = ReducirTemperatura(t_k, k);
      k = k + 1;
    until terminacion
  end
```

Tabla 5.2: Recocido Simulado

dos fases: una *fase constructiva* y una *fase de mejora*. En el algoritmo 5.3 se muestra el pseudo-código de alto nivel para esta metaheurística.

```
  {x: TipoSolucion} = GRASP(M: integer; f: TipoFuncionObjetivo)
var
  i: integer;
  x_trial: TipoSolucion;

  begin
    for i = 1 to M do
      {x_trial} = ConstruccionAleatorizadaMiope();
      {x_trial} = MejorarSolucion(x_trial, f);
      {x} = ActualizarSolucion(f, x, x_trial);
    end for
  end
```

Tabla 5.3: GRASP

La fase constructiva es un procedimiento iterativo encargado de construir una solución elemento a elemento. Inicialmente se parte de una semilla que es una componente o un conjunto de componentes que determinan una solución parcial. Las componentes introducidas en ella se señalan como elementos no seleccionables. El resto de componentes constituyen el *conjunto de elementos seleccionables*. Posteriormente, se ordenan todos los elementos seleccionables, utilizando para ellos una función *voraz*, *constructiva* o *miope* (*greedy*), que les asigna un coste relacionado con el cambio que se produce en la función objetivo si se introduce cada uno de los elementos en la solución parcial. Esta fase del algoritmo es la que le aporta el nombre de *Greedy*.

Una vez que se tienen ordenados todos los elementos seleccionables, se plantea el problema de elegir uno «*bueno*». En el contexto de GRASP no se selecciona el mejor candidato posible, ya que esta opción no asegura que se obtenga una solución óptima, sino que se elige aleatoriamente un candidato de un conjunto de buenos candidatos. Este conjunto recibe el nombre de *lista de candidatos restringida* o *RCL* (*Restricted Candidate List*). Numéricamente, la RCL se construye utilizando los valores máximo y mínimo del coste asignado a los elementos seleccionables en una iteración dada. Si se supone que c_{max} y c_{min} son respectivamente los valores más alto y más bajo del coste RCL estaría formada por todos aquellos elementos cuyo coste superase (para problemas de maximización) el umbral dado por la siguiente expresión:

$$RCL_{umbral} = (c_{min} + \alpha \times (c_{max} - c_{min}))$$

donde el parámetro $\alpha : 0 \leq \alpha \leq 1$ determina el tamaño de la RCL. Si $\alpha = 1$ en la RCL sólo estaría el mejor candidato (función miope pura). Por contra, si $\alpha = 0$, estarían todos

los candidatos (función aleatoria pura). En implementaciones estándares de GRASP el parámetro α se determina de forma aleatoria. Se han propuesto diversas estratégias para elegir el valor de α, entra las que destacan:

1. Seleccionar su valor aleatoriamente de acuerdo a una distribución uniforme de probabilidad discreta (caso general).
2. Auto-ajustar su valor según la calidad de las soluciones recientes obtenidas (*GRASP reactivo*).
3. Seleccionar su valor de una distribución no uniforme de probabilidad discreta decreciente (mayor probabilidad para los mejores valores).
4. Fijar su valor a un número concreto (próximo a la elección miope pura).

En la figura 5.1 se muetra gráficamente el aspecto que tendría una RCL. El tamaño de la RCL está marcado por la línea punteada gruesa, que naturalmente depende del valor α, pudiendo pasar de la elección puramente aleatoria a la elección puramente miope. Resaltar que esta fase es la que aporta la palabra *Randomized*.

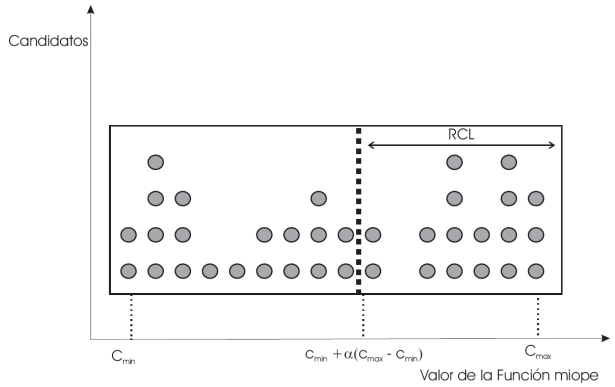


Figura 5.1: Lista de candidatos restringida para problemas de maximización.

Una vez que se ha seleccionado un candidato perteneciente a la RCL, éste se introduce en la solución parcial y se marca como elemento no seleccionable. El resto de elementos siguen siendo seleccionables; por consiguiente, para ellos se calcula de nuevo (mediante la función miope) la variación que se produciría en la función objetivo si se seleccionase cada uno de los elementos. Por tanto, según se van introduciendo candidatos a la solución parcial GRASP, ésta se va adaptando al nuevo escenario. A esta fase del procedimiento se debe el calificativo de *Adaptative*.

La fase constructiva termina cuando se dispone de una solución factible. En el algoritmo 5.4 se muestra un pseudo-código de alto nivel para esta fase.

La solución construida en la primera fase no tiene por qué ser un óptimo local, ya que existen bastantes elecciones estocásticas. En otras palabras, la fase constructiva no

```
  {x: TipoSolucion} = ConstruccionAleatorizadaMiope()
var
  p_x: TipoElemento; //Elemento de la solución
  L, RCL: list of TipoSolucion; //Lista y lista restringida de candidatos
  C: list of real; //Coste para cada candidato
  α: real;

  begin
    {L} = CandidatosSeleccionables();
    {C} = CalcularCoste(L);
    repeat
      {α} = rand(1);
      {RLC} = ConstruirRCL(L, C, α);
      {p_x} = Seleccionar(RCL); // Seleccionar un candidato aleatoriamente de RCL
      {L} = ActualizarCandidatosSeleccionables(L, p_x);
      {x} = Incluir(x, p_x);
      {C} = CalcularCoste(L);
    until EsFactible(x)
  end
```

Tabla 5.4: Fase constructiva de GRASP

garantiza una optimalidad de la solución con respecto a una estructura de vecindad dada. Para resolver este problema, GRASP introduce una segunda fase, conocida como *fase de mejora*, que consiste en un procedimiento de optimización local basado en una función de búsqueda local o, incluso, en una metaheurística. Por lo general, esta fase mejora la solución construida pero tampoco garantiza la optimalidad de la solución (aunque experimentalmente se ha comprobado que mejora bastante la calidad).

En la implementación estándar de GRASP, la fase de mejora es un procedimiento de búsqueda local; por lo tanto, habría que definir una estructura de vecindad, examinarla y hacer un movimiento hacia el vecino que produzca alguna mejora en la función objetivo. Este movimiento debe mantener intacto el hecho de que la solución sea factible. El proceso de búsqueda local se mantiene hasta que no se puede encontrar una solución mejor en la vecindad utilizada.

Enumeramos los factores determinantes que influyen en la búsqueda local:

- Estructura de vecindad (suele ser sencilla).
- Algoritmo de optimización local para esa vecindad (usualmente basado en un procedimiento de búsqueda local). En función del tipo de movimiento, los métodos de mejora se clasifican en:

  Primer movimiento de beneficio (*First-Improvement*): seleccionar el movimiento que conduzca al primer vecino que mejore la solución actual.

Mejor movimiento de beneficio (*Best-Improvement*): evaluar todos los movimientos hacia los vecinos y seleccionar aquel que produzca mayor beneficio.

En la práctica, las dos alternativas suelen conducir a la misma solución; por lo tanto, puesto que el segundo método tiene asociado un coste computacional más elevado, se suele utilizar el primero. Además, se ha observado empíricamente que con la segunda opción, GRASP converge con mayor probabilidad a óptimos no globales.

- Evaluación de la función de coste de los vecinos (GRASP utiliza vecindades pequeñas si la evaluación es muy costosa).

- De la propia solución inicial (GRASP construye soluciones de alta calidad para minimizar, en la medida de lo posible, este hecho).

La segunda fase termina cuando no se puede hacer ningún movimiento que mejore la solución actual. En el algoritmo 5.5 se muestra un pseudo-código de alto nivel para esta fase.

```
{x : TipoSolucion} = MejorarSolucion(x: TipoSolucion; f: TipoFuncionObjetivo)
var
  vecindad: array [1..N_V] of TipoSolucion;
  newSolucion: boolean;

  begin
    repeat
      newSolucion = FALSE;
      {vecindad} = GenerarVecindad(x);
      {x, newSolucion} = ActualizarSolucion(f, x, vecindad);
    until not newSolucion
  end
```

Tabla 5.5: Fase de mejora de GRASP

Existen una serie de métodos que se pueden introducir en la implementación estándar de GRASP que, para algunos problemas, pueden conseguir mejoras sustanciales. A continuación, se describen esquemáticamente los que se consideran de mayor interés:

- *GRASP reactivo*: este método consiste en dotar a GRASP de memoria, de forma que la elección de α no sea aleatoria, sino que tenga en cuenta la historia pasada. En este marco, tendrían más posibilidades de ser seleccionados aquellos valores de α que hayan conducido a soluciones de alta calidad en el pasado. En general, esta implementación mejora los resultados estándares de GRASP.

- *Perturbación de costes*: este método consiste en añadir un ligero ruido a los costes de forma similar a los *métodos ruidosos*. Esta opción añade flexibilidad a la implemetanción de GRASP, sobre todo en aquellos problemas que no son muy sensibles a la aleatorización. También es útil cuando no se dispone de una función miope que se pueda aleatorizar.

- *Funciones de desplazamiendo (bias functions)*: esta técnica establece un criterio de selección de candidatos en la RCL más inteligente, de forma que, en lugar de ser todos los candidatos equi-probables, se utiliza una función de distribución que enfatiza a unos candidatos sobre otros.

- *Construcción inteligente (memoria y aprendizaje)*: Este método consiste en la introducción de una memoria a largo plazo en el esquema GRASP, de tal forma que se tenga en cuenta la historia pasada a la hora de tomar una decisión. Originalmente fue propuesto por Fleurent y Glover como una estrategia muy útil para todas las metaheurísticas multi-arranque.

- *POP en construcción*: Las imperfecciones introduccidas durante la fase constructiva pueden ser «*limadas (ironed-out)*» aplicando un procedimiento de búsqueda local durante la fase de construcción. Debido a la relación que se establece entre la eficiencia y la calidad no se suele aplicar en todos los puntos.

6. ETIQUETADO LINEAL MÍNIMO

6.1 Descripción

El problema del *Etiquetado Linear Mínimo* (MINLA) fué anunciado la primera vez por Harper en su artículo *Optimal assignment of numbers to vertices*. Su objetivo fue el diseño error-corrección de códigos con una mínima media de errores absolutos en ciertas clases de grafos. Después, en los años 1970 MINLA fue usado como un modelo abstracto en el diseño de circuitos VLSI, dónde los vértices de un grafo representaban módulos y las aristas representaban interconexiones. En este caso, el coste de la longitud de los trazos de cable surgió a su vez en otros campos de estudio como en aplicaciones biológicas, dibujos de grafos, software de dibujo de diagramas y planificaciones en el trabajo.

El problema MINLA puede ser descrito formalmente como a continuación. Dado $G(V, E)$ un grafo finito unidireccional, donde $V(|V| = n)$ define el conjunto de vértices y $E \subseteq V \times V = \{\{i, j\} : i, j \in V\}$ es el conjunto de aristas. Dado una-a-una una función para la etiquetación de los vértices $\varphi : V \to \{1..n\}$, el total del coste de sus aristas para G con respecto al etiquetado φ está definido acuerdo a:

$$LA(G, \varphi) = \sum_{(u,v)\in E} |\varphi(u) - \varphi(v)|$$

Así pues, el problema MINLA consiste en encontrar la mejor función de etiquetado φ para un grafo G dado para que $LA(G, \varphi)$ sea minimizado.

Existen algoritmos polinónicos de tiempo exacto para algunos casos especiales de MINLA como árboles, ROOTED TREES, hipercubos, mallas, OUTERPLANAR GRAPHS, y otros. Aunque, como es normal en el resto de grafos, encontrar el etiquetado linear mínimo es un problema NP-duro. Así pues, nos vemos en la necesidad de crear heurísticas para este problema en tiempos razonables. Entre los algoritmos que podemos encontrar en la actualidad hay: a) heurísticas especialmente diseñada para el MINLA, como el *Improved Frontal Increase Minimization heuristic* (Arreglo Frontal para la Mejora de la Minimización), el *Binary Decomposition Tree heuristic* (Descomposición del Árbol Binario), el *Multi-Scale algorithm (SA)* (algoritmo multi-escalado) y el *Algebraic Multi-Grid scheme* (esquema algebraico multi-red); y b) metaheurísticas como el *Simulated Annealing (SA)*, *Memetic Algorithms* o *GRASP*

6.2 Métodos constructivos

La heurística propuesta por McAllister usa un algoritmo que pretende etiquetar los vértices más próximos entre ellos. Su propuesta consta de dos pasos esenciales:

1. Elegir un primer vértices al azar por el cual se comenzará y etiquetarle 1. Se irán etiquetándo los vértices de 1..n.

2. De los vértices que tenemos etiquetados, elegir sus adyacentes y comprobar el número de adyacentes no etiquetados que éstos tienen.

3. Elegir el que menos adyacentes no etiquetados tenga y etiquetarlo con la etiqueta i.

De esta manera, lo que se pretende conseguir es que los nodos queden concentrados y las etiquetas adjudicadas no se extiendan por todo el grafo, pues así los últimos nodos que se etiquetaran producirían una reacción en cadena en la que la diferencia a los demás sería muy grande.

El algoritmo hace uso de $sf(v) = d_U(v) - d_L(v)$, dónde $d_U(v)$ representa los vértices de v no etiquetados y $d_L(v)$ los vértices ya etiquetados. Mediante esta fórmula podemos obtener valores negativos, los cuales nos indican la urgencia de ese nodo a ser etiquetado. Los vértices ya elegidos no necesitarán la actualización de $sf(v)$ mediante esta fórmula, únicamente los nodos adyacentes a los nuevos elegidos se deberán restar 2 a su valor (figura 6.1).

Así pues dado L el conjunto de los vértices ya etiquetados y U el conjunto de vértices no etiquetados mostramos el algoritmo de McAllister (1999) que superó a los previos metodos constructivos.

Inicializar $L = \oslash$ y $U = V$
Seleccionar nodo u aleatoriamente de U.
Asignar etiqueta $l = 1$ a u. $L = \{u\}$, $U = U - \{u\}$
while $(U \neq \oslash)$
    $l = l + 1$
    Construir $CL = \{v \in U/(w, v) \in E \forall w \in L\}$
    Construir $sf(v) = d_U(v) - d_L(v)$ para todo v en CL
    Seleccionar vértice u en CL con mínimo $sf(u)$
    Etiquetar u con la etiqueta l
    $U = U - \{u\}$, $L = L \cup \{u\}$
end while

Tabla 6.1: Pseudo-código del método constructivo C1 basado en McAllister.

El segundo constructor propuesto es una derivación de C1. Proponemos un constructor basado en GRASP. En la implementación que a continuación usaremos, elegiremos los

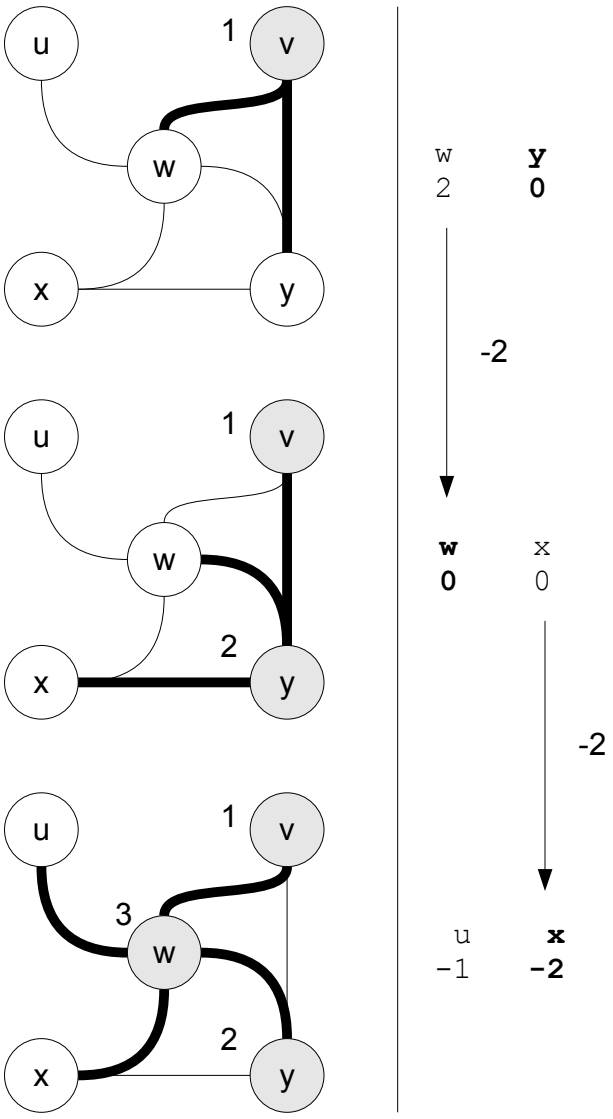


Figura 6.1: Restando menos dos a cada $sf(v)$.

candidatos a pertenecer en la lista RCL en función de la calidad de estas soluciones. La calidad la calcularemos mediante el valor de th. Así pues, los candidatos que superen o igualen el valor de th entrarán en la lista restringida. Este valor lo calcularemos en base al valor mínimo calculado por un nodo al asignarle una etiqueta

$$min_{v \in CL} sf(v)$$

y al valor máximo

$$max_{v \in CL} sfv$$

de la lista de candidatos seleccionables.

En la tabla 6.2 podemos ver el constructivo utilizado. Podemos ver como utilizamos el método constructivo de GRASP.

Inicializar $L = \oslash$ y $U = V$
Seleccionar nodo u aleatoriamente de U.
Asignar etiqueta $l = 1$ a u. $L = \{u\}$, $U = U - \{u\}$
while $(U \neq \oslash)$
    $l = l + 1$
    Construir $CL = \{v \in U / (w, v) \in E \forall w \in L\}$
    Construir $sf(v) = d_U(v) - d_L(v)$ para todo v en CL
    Construir $RCL = \{v \in CL / sf(v) \leq th\}$
    Seleccionar vértice u en RCL
    Etiquetar u con la etiqueta l
    $U = U - \{u\}$, $L = L \cup \{u\}$
end while

Tabla 6.2: Pseudo-código del método constructivo C2 basado en GRASP.

Siendo

$$th = msf + \alpha(Msf - msf),$$

$$msf = min_{v \in CL} sf(v),\ Msf = max_{v \in CL} sf(v)$$

El método constructivo C2 se diferencia de C1 en la manera de recoger los elementos de la lista seleccionable. En C1 siempre se selecciona el elemento con mínimo $sf(u)$ mientras que en C2 no nos guiamos por el «*mejor*» componente, sino que creamos una lista de candidatos en la que no siempre recogeremos el de menor $sf(u)$.

Por último, crearemos una nueva modificación utilizando de nuevo GRASP. En este caso usaremos la función llamada C para recalcular la contribución de cada vértice de la lista de candidatos a la solución. Seguidamente calcularemos th_c de la misma forma que en C2 pero observando únicamente la contribución. Así pues, tendremos que siendo

$C(v,l)$ la contribución de v, cuando v es etiquetado con la etiqueta l, a la solución actual:

$$C(v,l) = \sum_{u \in N(v) \cap L} |f(v) - l|$$

La tabla 6.3 muestra el pseudo-código del constructivo.

Inicializar $L = \oslash$ y $U = V$
Seleccionar nodo u aleatoriamente de U.
Asignar etiqueta $l = 1$ a u. $L = \{u\}$, $U = U - \{u\}$
while $(U \neq \oslash)$
    $l = l + 1$
    Construir $CL = \{v \in U / (w,v) \in E \forall w \in L\}$
    Construir $sf(v) = d_U(v) - d_L(v)$ para todo v en CL
    Construir $CLmsf = \{v \in CL / sf(v) = msf\}$
    Construir $RCL = \{v \in CLmsf / C(v) \leq th_c\}$
    Seleccionar vértice u en RCL
    Etiquetar u con la etiqueta l
    $U = U - \{u\}$, $L = L \cup \{u\}$
end while

Tabla 6.3: Pseudo-código del método constructivo C3 basado en GRASP.

El parámetro th_c en C3 establece los vértices con una relativa pequeña contribución a la solución actual. Esto puede ser computado como un porcentaje β de su rango en la lista de candidatos $CLmsf$:

$$th_c = dm_L + \beta(dM_L - dm_L),$$

$$dm_L = min_{v \in CLmsf} C(v,l),\ dM_L = max_{v \in CLmsf} C(v,l)$$

Si el parámetro β en C3 toma el valor 1, th_c equivale a dM_L, y todos sus vértices en $CLmsf$ están en RCL, existirá una elección aleatoria entre los vértices. Por otro lado, si toma el valor de 0, th_c equivale a dm_L, habrá una selección miope (*greedy*).

Hemos creado otros dos métodos constructivos que se basan en la elección del vértice empeorando (C4) o mejorando (C5) la solución actual. Este tipo de constructivo se basa en la elección de un vértice inicial al azar y se le etiquetará con 1. Seguidamente se irán etiquetando los restantes vértices de 2 a n, siendo n el número de vértices del grafo. El constructivo que *mejora* la solución, seleccionará los vértices utilizando los adyacentes de los ya etiquetados. Una vez seleccionados estos adyacentes, se les aplicará la etiqueta correspondiente y se comprobará cual es el que mejor aportación ofrece, el cual será elegido para etiquetarse. En el caso del constructivo que empeora la solución, éste elegirá aquellos nodos que más peor aportación ofrezcan a la solución.

En un principio, se puede considerar que el algoritmo que mejor aportaciones ofrece a la solución, el constructivo C4, es el constructivo ideal, pues va ofreciendo un etiquetado «*mejor*» a la solución final. Empíricamente hemos demostrado que esta solución ofrece los peores resultados. Ésto se debe a que en un principio se etiquetan los vértices aportando diferencias de 1 en 1. El grafo recorre, a modo de Dijkstra, el grafo intentando trazar una ruta en la que todavía no existan vértices etiquetados. Una vez que un vértice queda atrapado y no existen más adyacentes a él, la lista restante de los vértices adyacentes que aun no estaban etiquetados volverá hacia atrás y se etiquetará uno de sus nodos. Al etiquetar éste, se producirá una reacción en cadena en la que todos los nodos adyacentes a él tenían valores muy pequeños comparados con el actual.

No obstante, al contrario que el constructivo C4 que intentaba obtener la mejor solución, el constructivo que intenta dar la peor solución, C5, es mucho mejor que el C4. Ésto se demuestra empíricamente observando que los vértices intentan etiquetar a sus adyacentes cercanos como prioridad. De esta forma, los nodos se irán agrupando poco a poco y las diferencias entre nodos serán constantes y no se producirá una reacción en cadena como la del constructivo que intentaba mejorar.

Se puede ver el funcionamiento de estos dos constructivos en el pseudo-codigo 6.4.

```
n = ElegirNodoAlAzar();
AnadirALaSolucion(n, 1);
ListaAdyacentes = AnadirAdyacentes(n, ListaAdyacentes);
for i = 2 to nodosDelGrafo do
  n = ElegirNodoMenorAporte(ListaAdyacentes);
  // n = ElegirNodoMayorAporte(ListaAdyacentes);
  AnadirALaSolucion(n, i);
  ListaAdyacentes = QuitarNodo(n);
  ListaAdyacentes = AnadirAdyacentes(n);
end for
```

Tabla 6.4: Pseudo-código del método constructivo de mejoramiento (C4) y empeoramiento (C5).

6.3 Médodos de mejora

Se utilizó la metaheurística trayectorial *Recocido Simulado* (*Simulated Annealing*). Para utilizar este tipo de metaheurística, se tuvo que implementar dos tipos de operaciones: un generador de soluciones y una función de enfriamiento. Este tipo de heurística es fácil de implementar, pero la elección de los parámetros es esencial y puede generarnos diferentes tipos de resultados. Normalmente se utiliza una temperatura inicial alta e independiente de la solución. Las primeras soluciones generadas podrán hacer una elección entre soluciones buenas o malas con prácticamente la misma probabilidad. Más tarde, la temperatura se

irá ajustando para obtener mejores resultados y no combinar *malos candidatos* con una solución *buena* construida previamente.

Se implementaron unos métodos llamados *FlipN* y *FlipE* para seleccionar una solución. El principio de *FlipN* es seleccionar un número predeterminado de vértices en el grafo e intercambiarlos entre ellos. Estos vértices pueden ser adyacentes entre ellos o no, es decir, se seleccionan aleatoriamente. En el caso de elegir dos vértices, se intercambiarán entre ellos. Siendo $LV = u_0, ..., u_n$ una lista con 3 o más nodos, se intercambiará el nodo u_i $(i \neq 0)$ con el nodo u_{i+1}. El último nodo u_n se intercambiará con el primero u_0. *FlipN* nos proporciona un método que no deja atrapado el proceso en un mínimo local. Seguidamente SA decidirá sobre la selección del cambio respecto a su aportación. En el pseudo-código 6.5 podemos ver el funcionamiento de FlipN.

```
FlipN(numeroDeNodosAElegir : Entero)

LV = SeleccionarAleatoriamente(numeroDeNodosAElegir)
for i = 1 to i = numeroDeNodosAElegir do
  if i = numeroDeNodosAElegir then
    Intercambia(i, 0)
  else
    Intercambia(i, i + 1)
  end if
end for
```

Tabla 6.5: FlipN

El otro método utilizado en la selección de una solución es FlipE. FlipE consiste en recoger un vértice aleatoriamente de la solución y a continuación recoger uno de sus vértices adyacentes. De esta forma haremos un intercambio en los vértices cambiando el grafo únicamente en una de las zonas. Este tipo de cambio puede ser interesante en algunos tipos de grafos en los que existen vértices con solo un adyacente, ya que podremos situar las etiquetas que peor aportes nos den a la solución en las esquinas de éstos vértices. Así pues, éstos vértices etiquetados en los extremos no proporcionarán un aumento en los vértices vecinos. Podemos ver un ejemplo de este tipo de grafos en la figura 6.2.

En la elección del enfriamiento de la temperatura, se implementaron las funciones de la tabla 5.1 vista con anterioridad. Así pues, los métodos que se implementaron fueron:

- T1: Esquema de Cauchy.
- T2: Descenso geométrico.
- T3: Criterio de Boltzmann ($\alpha = 0{,}90$).
- T4: Lundi y Mees ($\beta = 0{,}01$).

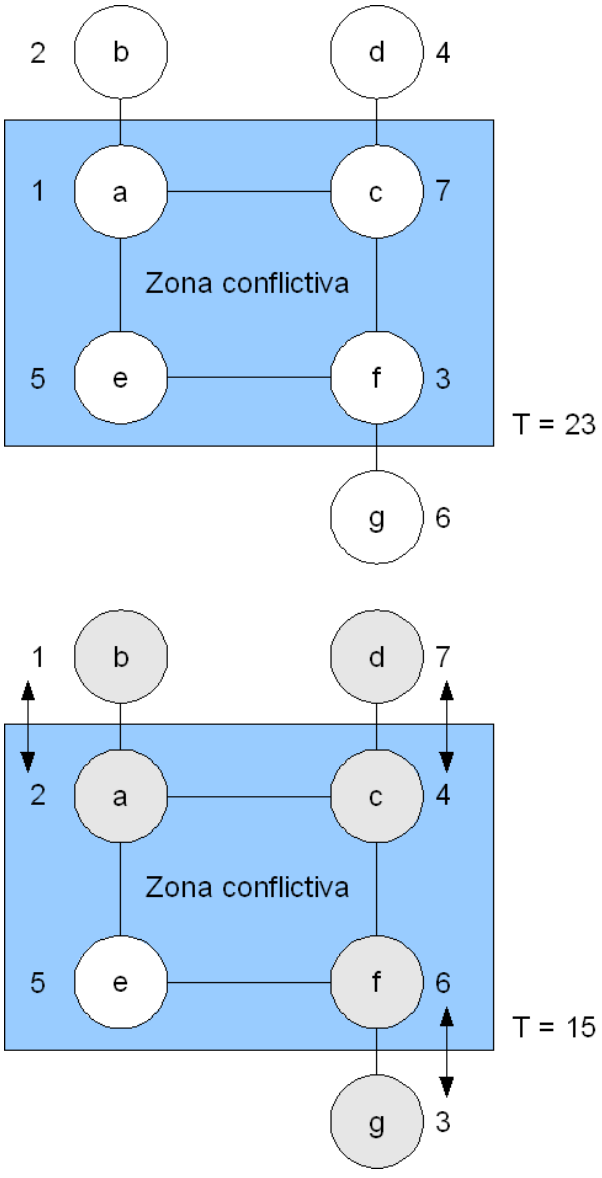


Figura 6.2: Uso de FlipE en un grafo.

6.4 Pruebas y resultados

Durante esta sección describiremos los experimentos computaciones que hemos realizado para comparar nuestras metaheurísticas para resolver el problema del Etiquetado Lineal Mínimo. Todos los experimientos fueron realizados en un ordenador personal 1.7 GHz Intel Centrino con 2.0 GB de RAM. El lenguaje de programación será $C\sharp$ bajo .NET Framework. Usaremos el conjunto de instancias de Petit.

Instancias de Petit: Conjunto de veintiuna instancias ($62 \leq n \leq 10,240$ y $125 \leq 30,380$) nombradas en Petit (2003a). Estas instancias fueron diseñadas para ser complicadas, es decir, no pueden ser resueltas mediante algoritmos de fuerza bruta. Este conjunto incluye grafos aleatorios y tres familias de grafos reales: VLSI, dibujar un grafo e ingeniería (dinámica de fluidos y mecanismos estructurales).

Ya que la metaheurística del recocido simulado provee de una amplia gama de parámetros, durante esta sección diviremos los resultados dependiendo de los constructores, los métodos de mejora y otro tipo de argumentos.

En esta primera fase realizaremos un estudio sobre los constructivos ya implementados:

- C1: Constructivo basado McAllister, 6.1.
- C2: Constructivo basado en GRASP versión 1 6.2.
- C3: Constructivo basado en GRASP versión 2 6.3.
- C4: Constructivo de mejoramiento 6.4.
- C5: Constructivo de empeoramiento 6.4.

Así pues, se elegieron una serie de instancias de las instancias de Petit: bintree10.gra, c1y.gra, c4y.gra, gd95c.gra, hc10.gra, mesh33x33.gra, randomA1.gra y randomG4.gra. Los resultados se basan en repetir cada proceso constructivo durante cien iteraciones y elegir la que mejor resultados presente, en valor, nos proporcione. El tiempo se basa en en las cien iteraciones. Estos resultados podemos verlos en la tablas 6.6 y 6.7.

La principal razón de haber separado las dos tablas se debe a que la tabla 6.7 obtiene unos resultados de muchísima peor calidad que los de la tabla 6.6. De esta manera, podemos comparar en la tabla 6.6 los constructores con mejores resultados.

Se han señalado en la tabla 6.6 en negrita los mejores resultados de cada constructivo. Como se pudo observar, la mayoría de los mejores resultados en valor se obtienen mediante el constructivo C1. En mejores resultados, le sigue muy de cerca el constructivo C3, con el cual se obtuvieron muy buenos resultados. Aunque en lo que a tiempo se refiere, se observó que el constructivo destaca incluso por varios segundos.

Los resultados obtenidos mediante los constructivos C4 y C5 destacan por los resultados de peor calidad que se obtienen. Llegando incluso a destacar por sus resultados de quince

	C1		C2		C3	
	valor	tiempo	valor	tiempo	valor	tiempo
bintree10.gra	45889	0.3204608	6696	0.9012960	**6401**	0.8712528
c1y.gra	**70865**	0.3204608	71038	1.2317712	71158	1.3619584
c4y.gra	141415	0.5608064	**136784**	2.0629664	138925	2.3233408
gd95c.gra	663	0.0100144	697	0.0400576	**590**	0.0500720
hc10.gra	**523776**	2.1430816	**523776**	6.3391152	**523776**	6.8598640
mesh33x33.gra	**33966**	0.2403456	**33966**	1.1816992	**33966**	1.2818432
randomA1.gra	**969570**	2.3133264	981402	6.5293888	977751	7.0000656
randomG4.gra	**290469**	0.7410656	291904	2.2932976	310909	2.5937296

Tabla 6.6: Resultados de los constructores para el problema *MINLA* (1).

	C4		C5	
	valor	tiempo	valor	tiempo
bintree10.gra	108601	0.5407776	210801	0.5407776
c1y.gra	1098055958	1.5422176	1114713258	1.1316272
c4y.gra	1855066525	3.2646944	1880697025	1.9427936
gd95c.gra	350090	0.0200288	458390	0.0100144
hc10.gra	367545776	2.9342192	554077176	2.3333552
mesh33x33.gra	41022366	1.7625344	44453166	0.2904176
randomA1.gra	581779751	3.0343632	821251851	2.8741328
randomG4.gra	1426934309	3.7453856	1530090309	1.4120304

Tabla 6.7: Resultados de los constructores para el problema *MINLA* (2).

mil veces peores de los mejores resultados obtenidos mediante los constructivos C1, C2 y C3. Aun así, se puede observar que los tiempos de éstos son muy similares a los mejores que se han obtenido.

De esta forma, se continuó el estudio de los resultados eligiendo el constructivo que mejores resultados nos proporcionó. Elegiremos pues, el constructivo C1.

Las pruebas que a continuación se realizarán se basaron en la elección de candidatos y en los tiempos de enfriamiento. En primer lugar se usó la elección de candidatos *FlipN* y usaremos los restantes métodos de enfriamiento en combinación. A continuación se realizará un estudio similar con el método de elección de candidatos *FlipE*. A continuación se muestra un resumen de los métodos utilizados.

Los métodos de elección de candidatos son:

- FlipN: Elección de nodos aleatorios, 6.5.
- FlipE: Elección de nodos aleatorios adyacentes, 6.2.

Las funciones de enfriamiento que usamos también podemos verlas con más detalle en la tabla 5.1:

- T1: Esquema de Cauchy.
- T2: Descenso geométrico.
- T3: Criterio de Boltzmann ($\alpha = 0{,}90$).
- T4: Lundi y Mees ($\beta = 0{,}01$).

Puesto que *FlipN* admite un argumento que señala el número de nodos a intercambiar, se realizaron pruebas con el intercambio de dos, tres y cuatro nodos. Los resultados obtenidos podemos verlos en las tablas 6.8, 6.9 y 6.10. Las pruebas se realizando aplicando diez veces los métodos de mejora sobre un constructivo inicial y recogiendo la mejor solución de todas.

	T1		T2		T3		T4	
	valor	tiempo	valor	tiempo	valor	tiempo	valor	tiempo
bintree10.gra	126665	24.0445744	121679	24.0646032	121525	24.0345600	**118081**	24.0646032
c1y.gra	196832	15.0216000	194803	15.0216000	192634	15.0216000	**189624**	15.0816864
gd95c.gra	**2456**	15.0216000	2641	15.0216000	2494	15.0216000	2489	15.0516432
hc10.gra	843376	15.0216000	844802	15.0216000	845650	15.0216000	**840630**	15.0516432
mesh33x33.gra	296241	15.0216000	**289559**	15.0216000	292674	15.0216000	297965	15.0516432
randomA1.gra	1129685	15.0216000	**1126479**	15.0216000	1127579	15.0216000	1133527	15.0516432
randomG4.gra	915023	15.0216000	923883	15.0216000	**905377**	15.0216000	914774	15.0516432

Tabla 6.8: Resultados de las mejoras aplicando SA con FlipN = 2.

	T1		T2		T3		T4	
	valor	tiempo	valor	tiempo	valor	tiempo	valor	tiempo
bintree10.gra	188965	48.1091776	**186872**	48.0991632	188725	48.0991632	189817	48.0991632
c1y.gra	290826	30.0432000	**287077**	30.0732432	290845	30.0432000	294833	30.0732432
gd95c.gra	2740	30.0432000	**2570**	30.0732432	2855	30.0432000	2652	30.0732432
hc10.gra	1051986	30.0432000	1052160	30.0832576	**1043786**	30.0432000	1060378	30.0732432
mesh33x33.gra	463040	30.0432000	445286	30.0732432	456711	30.0432000	**444237**	30.0732432
randomA1.gra	1267736	30.0432000	1259117	30.0732432	**1254628**	30.0432000	1268446	30.0732432
randomG4.gra	1398232	30.1934160	1451877	30.0732432	**1385934**	30.0432000	1443022	30.0732432

Tabla 6.9: Resultados de las mejoras aplicando SA con FlipN = 3.

Al igual que en los resultados anteriores, se han resaltado en negrita los mejores resultados obtenidos. Se pudo comprobar en una primera observación que los mejores resultados con respecto a las tablas 6.8, 6.9 y 6.10 son obtenidos mediante el uso de *FlipN* = 2. Además se comprobó empíricamente que al aumentar el valor de *FlipN* no solo produce peores resultados, sino que el tiempo de ejecución se eleva.

	T1		T2		T3		T4	
	valor	tiempo	valor	tiempo	valor	tiempo	valor	tiempo
bintree10.gra	**244461**	1:12.14373	247439	1:12.13372	245961	1:12.13372	247297	1:12.13372
c1y.gra	352092	45.0948432	**350701**	45.0948432	352888	45.0948432	355559	45.0648000
gd95c.gra	**2546**	45.0948432	2834	45.0948432	2669	45.0948432	2727	45.0648000
hc10.gra	1171166	45.0948432	1175278	45.0948432	1181840	45.0948432	**1168838**	45.0648000
mesh33x33.gra	583618	45.7157360	**573030**	45.0948432	579245	45.0948432	581195	45.0648000
randomA1.gra	1386925	45.0948432	1376553	45.0948432	**1375293**	45.1048576	1384739	45.0648000
randomG4.gra	1879396	45.0848288	1881578	45.1048576	**1865838**	45.0948432	1905854	45.0748144

Tabla 6.10: Resultados de las mejoras aplicando SA con FlipN = 4.

Basándonos en el uso del cambio en las temperaturas y en las veintiuna pruebas obtenidas, se resumen los mejores resultados en la tabla 6.11. Se observó que el *Esquema de Cauchy* es el método que menos buenos resultados obtiene en total. Los resultados de los restantes métodos de enfriamiento, *Descenso geométrico*, *Criterio de Boltzmann* y *Lundi y Mess*, obtuvieron mayor cantidad en los resultados.

			Aciertos			
Id	Nombre del método	Argumentos	FlipN = 2	FlipN = 2	FlipN = 2	Total
T1	Esquema de Cauchy		1	0	2	3
T2	Descenso geométrico		2	3	2	7
T3	Criterio de Boltzmann	$\alpha = 0{,}90$	1	3	2	6
T4	Lundi y Mess	$\beta = 0{,}01$	3	1	1	5

Tabla 6.11: Comparación de los mejores resultados dependiendo del cambio en la temperatura y FlipN.

Así pues, se concretó que los mejores resultados usando $FlipN = 2$ y el constructivo $C1$ se obtuvieron aplicando los métodos de enfriamiento del *Descenso Geométrico* y el método de *Lundi y Mess* con $\beta = 0{,}01$.

A continuación se aplicará el estudio de los resultados para *FlipE*. *FlipE* intercambia dos nodos adyacentes entre ellos. Los resultados obtenidos se muestran en la tabla 6.12 y, al igual que con *FlipN* se usaron los diferentes criterios de enfriamiento. Se realizaron diez iteraciones por cada método del cual se obtuvo el mejor resultado.

Se pudo observar que los métodos del *Descenso Geométrico*, *Criterio de Boltzmann* con $\alpha = 0{,}9$ y *Lundi y Mess* con $\beta = 0{,}01$ son eficaces usando *FlipE*. Al igual que con las pruebas con *FlipN*, el *Esquema de Cauchy* no es muy efectivo en muchos de los grafos.

El tiempo de ejecución en los algoritmos para *FlipE* se reducen considerablemente pues tienen una complejidad lineal, al contrario de *FlipN* con complejidad exponencial.

La tabla 6.13 muestra la comparación de los mejores resultados obtenidos con el estado del arte actual.

	T1		T2		T3		T4	
	valor	tiempo	valor	tiempo	valor	tiempo	valor	tiempo
bintree10.gra	45289	0.0801152	43993	0.0801152	**42503**	0.0801152	44955	0.0801152
c1y.gra	70865	0.0400576	**70565**	0.0400576	70865	0.0400576	70865	0.0500720
gd95c.gra	663	0.0300432	651	0.0300432	663	0.0200288	**642**	0.0300432
hc10.gra	524476	0.0400576	**523876**	0.0500720	524776	0.0400576	524884	0.0500720
mesh33x33.gra	33669	0.0500720	**33666**	0.0400576	**33666**	0.0500720	33966	0.0400576
randomA1.gra	**969270**	0.0400576	969570	0.0500720	969570	0.0400576	969570	0.0500720
randomG4.gra	290469	0.0500720	290469	0.0400576	290469	0.0400576	**289869**	0.0500720

Tabla 6.12: Resultados de las mejoras aplicando SA con FlipE.

	Valor	Dev.
bintree10.gra	42503	8.96
c1y.gra	70565	4.69
gd95c.gra	642	0.26
hc10.gra	523876	0.00
mesh33x33.gra	33666	0.02
randomA1.gra	969270	34.89
randomG4.gra	289869	0.61

Tabla 6.13: Comparación de los mejores resultados con el estado del arte actual.

6.5 Conclusiones

Se han desarrollado, para el problema de *Etiquetado Lineal Mínimo*, varias soluciones basándonos en la metaheurística de *Recocido Simulado*. Para poner en marcha esta técnica, se implementaron cinco métodos constructivos que nos aportaron una solución inicial al problema. Tres de estos métodos son variaciones del constructivo de McAllister basados en la técnica de creación de constructivos de la metaheurística GRASP. Los dos posteriores se basan en añadir uno a uno soluciones a la solución general, de forma que mejoren o empeoren la solución actual.

Otro de los requisitos de esta metaheurística fue la implementación modelos de enfriamiento y de actualización de la solución actual. Se implementaron cuatro fórmulas de enfriamiento de la temperatura basadas en ecuaciones ya existentes: esquema de Cauchy, descenso geométrico, criterio de Boltzmann y el método de Lundi y Mess. Además se construyeron dos métodos de intercambio llamados *FlipN* y *FlipE* que intercambian nodos al azar, sean adyacentes en uno de los casos o aleatoriamente en el otro.

Durante las pruebas generadas repasamos los métodos constructivos que mejor soluciones nos aportaron, así como los métodos de enfriamiento y de intercambio. Y obtuvimos resultados cercanos al estado del arte combinando nuestras mejores estratégias. Los mejores resultados recogidos durante esta investigación se han obtenido usando la metaheurística *Recocido Simulado* con los siguientes métodos:

Constructor de McAllister + FlipE + {Descenso geométrico | Criterio de Boltzmann ($\alpha = 0{,}9$) | Lundi y Mess ($\beta = 0{,}01$)}

7. PERFIL

7.1 Descripción

El problema del perfil fue descrito formalmente por Díaz et al. en 1991 y por Lin y Yuan en 1994. En esta definición se propone utilizarlo como forma para reducir el espacio de matrices. Este problema es equivalente al problema citado por Ravi llamado *Interval Graph Completion* en 1991, el cual tuvo aplicaciones arqueológicas y de detección de huellas dactilares.

El problema del Perfil (*Profile*) puede ser descrito formalmente como a continuación. El *perfil* de un etiquetado φ de G es:

$$PR(\varphi, G) = \sum_{u \in V} (\varphi(u) - \min_{v \in \Gamma^*(u)} \varphi(v)).$$

Así pues, el problema PROFILE consiste en encontrar la mejor función de etiquetado φ para un grafo G dado para que $PR(\varphi, G)$ sea minimizado. Éste problema tiene mucha similitud con el MinLA, pero a diferencia de éste, el perfil hace la diferencia absoluta de un vértice con el su adyacente con menor valor en su etiqueta. Podemos ver un ejemplo resuelto de perfil en la figura 7.1.

7.2 GRASP

Para la resolución de este problema optamos por la metaheurística GRASP. GRASP (*Greedy Randomized Adaptive Search Procedure*) es un procedimiento multiarranque en la que cada iteración se compone de dos fases: una *fase constructiva* y una *fase de mejora*. Nuestra propuesta será crear un constructor y varios tipos de mejoras.

El constructivo se encargará de construir una posible solución «*buena*» que se basará de los principios que comentaremos a continuación. La fase constructiva se basa en la selección vértice a vértice de la solución aportando el coste al incluirlo en ésta. Para la selección de uno de los vértices se creará una posible lista L llamada «*lista de candidatos*» en la que en cada iteracción en la elección de un vértice se actualizará. El criterio de la elección de esta lista dependerá de problema. A continuación se creará una sublista RCL llamada «*lista de candidatos restringida*» en la que se seleccionarán un subconjunto de vértices de L. El criterio de selección dependerá del usuario. No obstante, existen varios métodos en la selección de los vértices de esta lista:

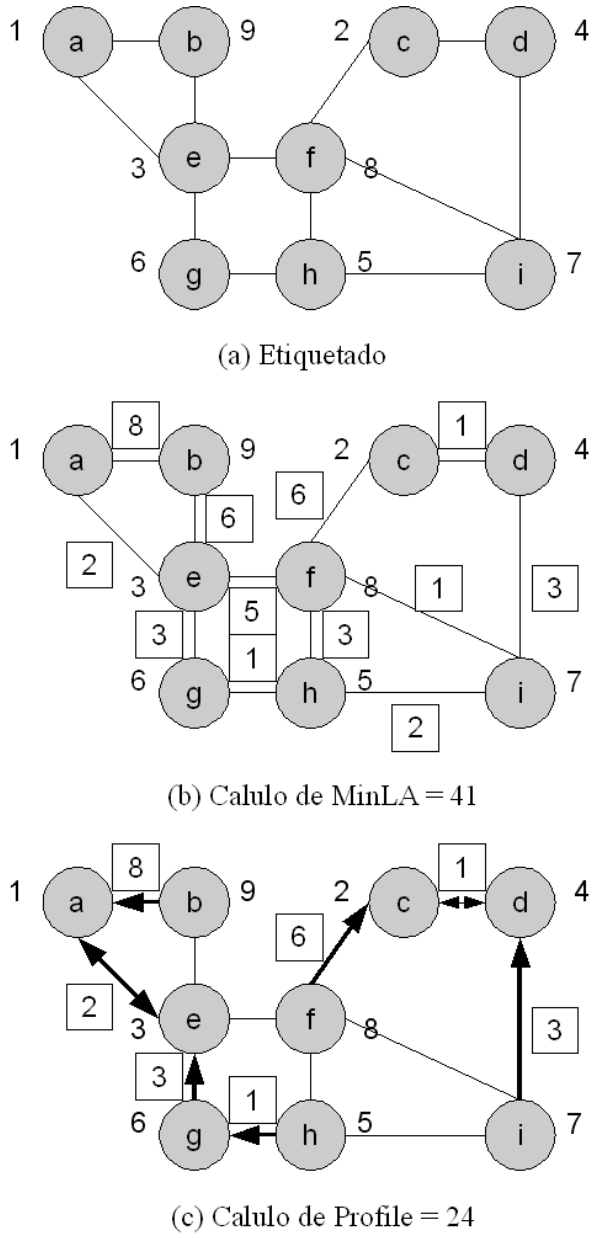


Figura 7.1: Comparación entre MinLA y Profile en un grafo.

- Elección de los mejores vértices relacionados con el coste que aplican a la solución.
- Elección de un número de vértices que apliquen un buen coste a la solución y otros que sean elegidos aleatoriamente.
- Elección aleatoria de los vértices.

Para la elección de los mejores vértices de la lista, usamos el criterio α visto anteriormente en 5.1. Nuestra implementación elege un número aleatorio entre 1 y en número de vértices de la lista L y éste es el número de vértices que contenga RCL. Además, se seleccionan los vértices que mejoren la solución actual, es decir, los que tengan mejor

coste. Matemáticamente, siendo $L = v_1, v_2, ..., v_n$ entonces $\alpha \in 1, ..., n$ y la lista $RCL \subseteq L$ y $|RCL| = \alpha$.

El pseudo-código para la elección de la lista restringida lo podemos ver en 7.1.

```
α = |L|;
L = OrdenarLista(); // Puede que ya la lista L ya esté ordenada
for i = |L| to |L| − α do
  RCL = Insertar(RCL, L_i);
end for
```

Tabla 7.1: Pseudo-código para la elección de la lista RCL.

Una vez construida la lista de candidatos restringida, se elige aleatoriamente uno de ellos y se incluye el vértice en la solución. El coste también se incrementa en la solución. Además deberemos recalcular el coste de los elementos de la lista L.

La actualización de la lista de candidatos se basa en la recogida de los vértices adyacentes al último elegido. De esta forma, la lista se incrementa poco a poco. Al ser un conjunto, en caso de que los adyacentes nuevos elegidos pertenezcan a éste, no se añaden. En caso de que un vértice no contenga adyacentes nuevos que no pertenezcan a la lista no se añade ninguno, pero la lista contiene los vértices anteriormente seleccionados. La actualización de la lista no quita los vértices anteriormente elegidos, solo añade nuevos.

Al tratarse del problema del *Perfil*, la actualización del coste depende del vértice adyacente que menor etiquetado tenga. Ya que nuestro procedimiento etiqueta desde $0..n$, siendo n el número de vértices del grafo G, se añade con valor 1 todos aquellos vértices adyacentes del nuevo introducido a la solución con valor 1, pues es la diferencia que se aplicaría en caso de ser elegido posteriormente uno de estos vértices. Además, se suma 1 al resto de vértices, pues al aumentar el número del etiquetado, ésta es la diferencia entre su antiguo vértice adyacente que lo seleccionó y el etiquetado de este en caso de ser elegido posteriormente. El pseudo-código de este procedimiento puede verse en 7.2.

El método de mejora en GRASP es un proceso de optimización local basado en una función de búsqueda local o una metaheurística. El método de mejora selecciona una vecindad en la que trabajar y optimizar. Ésta mejora finaliza en el momento en el que no se puede mejorar más la solución.

La mejora que se implementó funciona como se describe a continuación y a la cual llamaremos *Mejora por Flechas Virtuales*. Dado un grafo G ya etiquetado, se elige un vértice aleatorio v y se recoge el conjunto de vértices ϖ adyacentes a éste y que tengan como adyacente con mínima etiqueta el vértice v seleccionado. Se añade al conjunto ϖ el vértice v y se crea una lista ordenada con las etiquetas de los vértices del conjunto. Se elege la etiqueta κ que corresponderá con la mediana de la lista. Para finalizar, se intercambian el vértice correspondiente a la etiqueta κ con el vértice v.

CalcularCoste(p_x : integer; $label$: integer; C: **list of** integer)

$C = C - C_{p_x}$;
ady = ListaAdyacentes(p_x);

// Añadimos los nuevos
for i = 0 **to** $|ady|$ **do**
  $C_{|C|+i} = 0$;
end for

// Sumamos 1 a todos
for i = 0 **to** $|C|$ **do**
  $C_i = C_i + 1$
end for

Tabla 7.2: Pseudo-código para la actualización del coste.

A continuación podemos ver un ejemplo de lo anteriormente citado. Podemos ver en la figura 7.2 un ejemplo de grafo en el que existen flechas que señalan la elección de cada vértice con respecto a su adyacente con menor etiqueta.

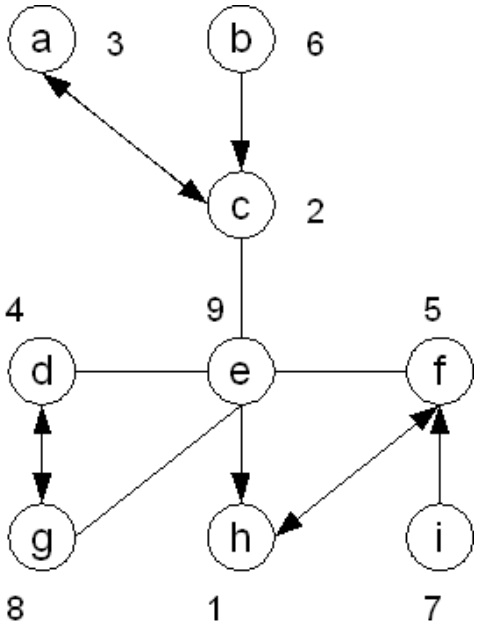


Figura 7.2: Ejemplo de mejora M1 para el perfil: Grafo inicial etiquetado.

A continuación elegiremos el vértice c y recogeremos los vértices adyacentes a él que le apuntan con una flecha, es decir, que su adyacente con menor etiquetado corresponde al vértice c, podemos ver la elección de estos vértices en la figura 7.3.

Como podemos ver, el resultado de perfil en esa zona equivale a:

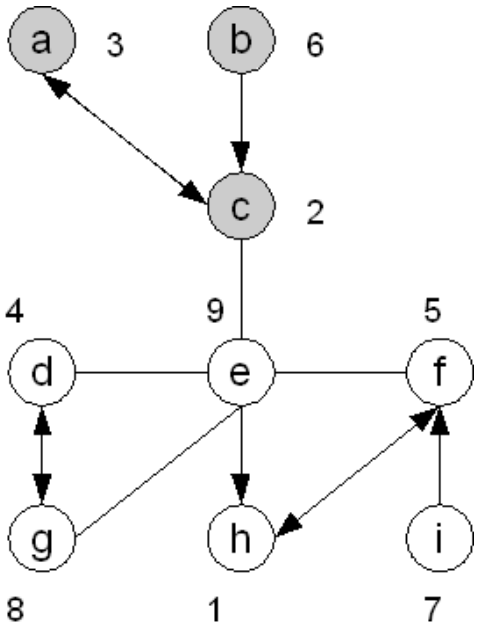


Figura 7.3: Ejemplo de mejora M1 para el perfil: Eligiendo vértices adyacentes.

$$|\varphi(a) - \varphi(c)| + |\varphi(b) - \varphi(c)| = 1 + 4 = 5$$

Recogiendo todos los vértices y añadiendo el seleccionado, c, crearemos una lista ordenada según el tamaño de la etiqueta:

$$\varpi = \{c, a, b\}$$

En la lista ϖ no debe entrar e, pues no apunta al vértice c seleccionado. Seguidamente, elegiremos el vértice que corresponde a la mediana de la lista ϖ y lo intercambiaremos por el vértice elegido c, como podemos ver en la figura 7.4.

Recalculando el valor de la zona, contemplamos una disminución en el valor:

$$|\varphi(a) - \varphi(c)| + |\varphi(b) - \varphi(c)| = 1 + 3 = 4$$

Este método ha sido creado por la siguiente razón: intentar no hacer cambios que afecten a un número grande de vértices, ya que al hacer un cambio de un vértice con otro, se produce una reacción en cadena en la que se ven afectados todos los vértices adyacentes de los vértices cambiados. Así pues, eligiendo únicamente los vértices adyacentes que apuntan al vértice aleatorio elegido, comprobamos como las flechas virtuales dibujadas en nuestro grafo no cambian de posición, ya que los vértices afectados se «*apuntan*» entre ellos. En el ejemplo anterior, en el caso de elegir el vértice «*e*» que no apunta al vértice elegido, no

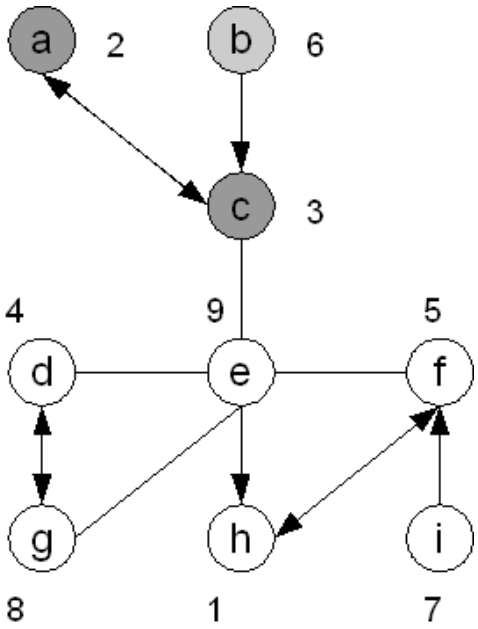


Figura 7.4: Ejemplo de mejora M1 para el perfil: Intercambiando los vértices elegidos.

podremos controlar los cambios que se podrán realizar en sus vértices adyacentes.

No obstante, este método no puede controlar al 100 % las reacciones en cadena que se puedan realizar al hacer un cambio, pero si que estas reacciones en cadena sucedan con menos frecuencia.

La implementación de nuestra mejora consiste en seleccionar un número aleatorio de nodos de la solución, entre el 4 % y 6 % del total de nodos de la solución, y realizar los cambios mediante el método anterior. En el caso de realizar los cambios y no producirse una mejora, se desecha ese cambio y se elige el vértice siguiente en la lista de los elegidos. En el pseudo-código 7.3 podemos ver el funcionamiento de la mejora.

```
MejoraM1(numVertices : integer;)

vertices = SeleccionaVerticesAleatorios(numVertices);
for i = 0 to |vertices| do
  solucionAuxiliar = CambiarVertice(vertices, solucionActual);
  if Coste(solucionAuxiliar) < Coste(solucionActual) then
    solucionActual = solucionAuxiliar;
  end if
end for
```

Tabla 7.3: Pseudo-código para la mejora M1 del perfil.

7.3 Pruebas y resultados

Durante esta sección describiremos los experimentos computaciones que se realizaron para comparar nuestras metaheurísticas para resolver el problema del *Perfil*. Todos los experimientos fueron realizados en un ordenador personal 1.7 GHz Intel Centrino con 2.0 GB de RAM. El lenguaje de programación será *C♯* bajo .NET Framework. Usaremos el conjunto de instancias de Petit.

Instancias de Petit: Conjunto de veintiuna instancias ($62 \leq n \leq 10,240$ y $125 \leq 30,380$) nombradas en Petit (2003a). Estas instancias fueron diseñadas para ser complicadas, es decir, no pueden ser resueltas mediante algoritmos de fuerza bruta. Este conjunto incluye grafos aleatorios y tres familias de grafos reales: VLSI, dibujar un grafo e ingeniería (dinámica de fluidos y mecanismos estructurales).

Así pues, se elegieron una serie de instancias de las instancias de Petit: bintree10.gra, c1y.gra, c4y.gra, gd95c.gra, hc10.gra, mesh33x33.gra, randomA1.gra y randomG4.gra.

Hemos realizado las pruebas implementando GRASP y el método de mejora anteriormente citado: Cambio de vértices dependientes de flechas virtuales. Podemos ver los resultados en la tabla 7.4. Las pruebas realizadas se basaron en la aplicación de GRASP diez veces y recogiendo la mejor solución proporcionada.

	Valor	Tiempo
bintree10.gra	6642	3.8956016
c1y.gra	23091	7.2504256
gd95c.gra	910	0.1101584
hc10.gra	400711	4:05.0824112
mesh33x33.gra	79424	29.3121488
randomA1.gra	393131	3:54.7675792
randomG4.gra	288484	2:53.1790192

Tabla 7.4: Resultados de GRASP para el problema del perfil.

Como observamos las soluciones obtenidas tienen gran calidad, esto es, valores muy bajos. En algunos de los casos de grafos muy aleatorizados los tiempos obtenidos se elevaron de gran manera con respecto a otro tipo de valores. Además este tipo de grafos contiene multitud de nodos. Las instancias de Petit ofrecen grafos con un número elevado de nodos. En implementaciones reales se puede observar como el tiempo se eleva exponencialmente según en número de nodos del grafo.

7.4 Conclusiones

Se han desarrollado, para el problema de *Perfil*, una solución basada en GRASP. Se debió adaptar esta metaheurística para poder resolver de forma eficaz este problema.

Para ello, se implementó métodos de elección de listas RCL baándonos en la calidad de los vértices disponibles, métodos óptimos para la recalculación del coste y mejoras basándose en la elección de vértices aleatorios

Se obtuvieron resultados de los cuales podimos concluir que sus resultados son eficientes. No obstante, se observó como estos métodos ofrecen un tiempo de ejecución dependiente del número de vértices, que afectan en gran medida al tiempo total de la aplicación.

El método usado durante la implementación fue:

GRASP: RCL basada en los vértices disponibles + Método de mejora basada en flechas virtuales

8. CONCLUSIONES Y TRABAJO FUTURO

8.1 Conclusiones

Durante la lectura de esta memoria, contemplamos un repaso histórico sobre la resolución de problemas de etiquetado lineal de grafos. Desde sus orígenes hasta la actualidad. Se repasan muchas de las aplicaciones que tiene este problema en la vida real: análisis numérico, circuitos VLSI, dibujo de grafos, procesamiento paralelo, etc. Se contempla una descripción formal para todos aquellos problemas que tienen como modelo la etiquetación de los nodos en un grafo, como el Perfil, el Corte de Suma, el Ancho de Banda, el Etiquetado Lineal Mínimo, etc. Y se hace un repaso general de la teoría de grafos.

El lector puede hacer un seguimiento cercano a algunas de las metaheurísticas existentes en la actualidad. En concreto, la metaheurística *Recocido Simulado* y GRASP. Así pues, implementaremos algunas de estas metaheurísticas en problemas reales de etiquetado de grafos como son el problema del Etiquetado Mínimo Lineal y el Perfil. Tanto estas técnicas como estos problemas se revisarán exaustivamente y propondremos unos algoritmos, basados en metaheurísticas, que resolverán estos problemas.

Finalmente, encontraremos unas soluciones cercanas al estado del arte. Además se harán observaciones de los resultados obtenidos y obtendremos algunas de las conclusiones finales.

8.2 Trabajo futuro

Existe un campo abierto en el estudio de la resolución de problemas de etiquetado lineal de grafos. Existen multitud de implementaciones que nos proporcionan soluciones óptimas en tiempo, pero aun existen una pequeña barrera entre el resultado óptimo y los resultados obtenidos.

Estos campos de estudio no solo se basan en la obtención de algoritmos para la resolución de grafos generales, también existen investigaciones para la aplicación de estrategias a tipos de grafos concretos. Existen multitud de grafos aun no resueltos en tiempos óptimos o que nos proporcionen soluciones óptimas. Otros de los resultados obtenidos con anterioridad poseen complejidades en tiempo bastante elevadas que son necesario aliviar en la actualidad.

Así pues, la resolución de problemas de etiquetado lineal son problemas que se llevan estudiando desde hace décadas y que aun siguen fascinando a los investigadores en la actualidad.

BIBLIOGRAFÍA

[1] P. Shope A. Cimikowski. *A neural-network algorithm for a graph layout problem.* 1996.

[2] Micael Gallego Carrillo Abraham Duarte Muñoz, Juan José Pantrigo. *Metaheurísticas.* Dykinson, S.L. - Libros, 2007.

[3] José Torres-Jiménez Eduardo Rodriguez-Tello, Jin-Kao Hao. *An Effective Two-Stage Simulated Annealing Algorithm for the Minimum Linear Arrangement Problem.* 2007.

[4] Arnold L. Rosenberg Fan R. K. Chung, Frank Thomson Leighton. *A layout problem with applications to VLSI design.* 1987.

[5] Yung-Ling Lai Gerard J. Chang. *On the profile of the corona of two graphs.* 2003.

[6] Maria Serna Josep Díaz, Jordi Petit. *A Survey of Graph Layout Problems.* 2002.

[7] Vicente Campos Rafael Martí Juan-José Pantrigo, Abraham Duarte. *Heuristics for the Minimum Linear Arrangement Problem.* 2007.

[8] D. F. Robinson L. R. Foulds. *Graph theoretic heuristics for the plant layout problem.* 1978.

[9] Toshiyuki Masui. *Evolutionary learning of graph layout constraints from examples.* 1994.

[10] Isabel Rojas Moritz Y. Becker. *A graph layout algorithm for drawing metabolic pathways.* 2001.

[11] Dorothea Wagner Ulrik Brandes. *A bayesian paradigm for dynamic graph layout.* 1997.

[12] Kenneth Williams Yung-Ling Lai. *A Survey of Solved Problems and Applications on Bandwidth, Edgesum and Profile of Graphs.* 1999.