TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Estudio e implementación de mecanismos de asignación de longitudes de onda para redes OBS sin colisiones.

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Telemática

AUTORES: Bárbara Morey Pascual
Marc Naharro Parra

DIRECTORA: Anna Agustí Torra

FECHA: 27 de enero de 2009

Título: Estudio e implementación de mecanismos de asignación de longitudes de onda para redes OBS sin colisiones.

Autores: Bárbara Morey Pascual
Marc Naharro Parra

Directora: Anna Agustí Torra

Fecha: 27 de enero de 2009

Resumen

En la última década, el tráfico en Internet ha crecido exponencialmente. Con el objetivo de dar respuesta a ésta demanda creciente, se ha favorecido la introducción de sistemas basados en el multiplexado por longitud de onda sobre redes de fibra óptica.

En ésta evolución hacia una Internet óptica, la técnica de conmutación tiene especial importancia. La conmutación óptica de ráfagas (*Optical Burst Switching*, OBS) se definió con el objetivo de aprovechar mejor los recursos de la red que utilizando la conmutación óptica de circuitos pero sin las exigencias tecnológicas que requiere la conmutación óptica de paquetes. En OBS los datos de los usuarios se agregan en la entrada de la red formando unidades de datos de mayor longitud, denominadas ráfagas. Antes de la transmisión de cada ráfaga, se transmite un paquete de control con el objetivo de configurar los dispositivos intermedios de la red para que la transmisión de la ráfaga se pueda realizar íntegramente en el dominio óptico. A menudo, la transmisión de una ráfaga se realiza sin tener confirmación sobre la disponibilidad de los recursos en todos los nodos intermedios de la red, con lo cual se pueden producir contiendas (contention) que, si no son resueltas, pueden suponer la pérdida de datos.

Se han propuesto varias técnicas para minimizar las pérdidas en redes OBS. En [19] los autores proponen una estrategia de transmisión libre de pérdidas basada en la combinación de un mecanismo de asignación de caminos y longitudes de onda, con esquemas básicos de resolución de contiendas. En éste esquema, para realizar la asignación de caminos y longitudes de onda, se clasifican las comunicaciones según su nivel de interferencia y se modela la red mediante un grafo cuyos vértices representan las comunicaciones y sus aristas, las interferencias. Sobre el grafo resultante, el problema de asignación de caminos y longitudes de onda es equivalente a un problema de coloreado de grafos.

En este trabajo se ha desarrollado una aplicación que permite realizar el coloreado de un grafo según diferentes algoritmos.

Title: Study and implementation of wavelength assignment mechanisms of Optical Burst Switching (OBS) networks without collisions.

Authors: Bárbara Morey Pascual
Marc Naharro Parra

Director: Anna Agustí Torra

Date: January, 27th 2009

Overview

During the last decade, Internet traffic has grown exponentially. To cope with this increasing traffic demand, systems based on the wavelength division multiplexing technique have been introduced in optical networks.

In this evolving process to an optical Internet, the switching technique has emerged as a key issue. The Optical Burst Switching (OBS) paradigm was defined to achieve more efficient resource utilization than using circuit switching, without the technological requirements of the optical packet switching paradigm. In OBS user data units are aggregated at the ingress nodes in larger data units, called bursts. Before the transmission of each burst, a control packet is sent in order to configure all intermediate nodes, so that the burst transmission can be performed transparently in the optical domain. Usually, burst transmissions start without confirmation regarding the resources availability in each intermediate link. Therefore, several bursts can compete for the use of the same resources, causing burst contention that, if not resolved, can result in data loss.

Several techniques have been suggested in order to minimize losses in OBS networks. In [19] the authors suggest a loss-free transmission strategy based on combining a routing and wavelength assignment technique with simple contention resolution schemes. In such a strategy, communications are classified according to their level of interference. A graph is used to represent communications (vertices) and the interferences among them (arcs). Then, the problem of assigning a path and a wavelength to each communication is equivalent to find a suitable vertex-colouring of such a graph.

In this work, an application that executes the vertex-colouring of the graph using several graph colouring algorithms has been developed.

A la Anna, por habernos dado la oportunidad de realizar este trabajo.

A ella misma agradecerle también las horas dedicadas a ayudarnos.

A las familias, por el apoyo y la comprensión ofrecidos.

A los profesores que nos han aportado su granito de sabiduría.

A los amigos y compañeros de clase por apoyarnos y darnos consejo.

A todos ellos muchas gracias.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La conmutación óptica de ráfagas (*Optical Burst Switching, OBS)* es una técnica de conmutación definida para incrementar la utilización de los recursos de la red en relación a la conmutación óptica de circuitos, pero sin los requisitos tecnológicos que exige la conmutación óptica de paquetes. En una red OBS, el plano de datos está separado del plano de control. En los nodos de entrada de la red, las unidades de datos de los usuarios se agregan formando unidades de datos de longitud superior, denominadas ráfagas. Antes de la transmisión de cada ráfaga, se transmite un paquete de control que es procesado electrónicamente en cada nodo intermedio con el objetivo de configurar las matrices de conmutación de los conmutadores de la red, permitiendo así que la ráfaga se transmita de forma transparente en el dominio óptico.

Para reducir los retardos de la red, se suelen utilizar mecanismos de señalización unidireccionales (*one-way*), en los que la transmisión de cada ráfaga se inicia sin conocer la disponibilidad de los recursos solicitados en todos los enlaces intermedios de la red. Así pues, pueden producirse contiendas cuando dos o más ráfagas solicitan el uso simultáneo de la misma longitud de onda en algún enlace intermedio de la red.

Se han propuesto varios mecanismos de resolución de contenciones para minimizar la pérdida de datos en redes OBS. Entre ellos, hay una estrategia sin pérdidas que garantiza el éxito de la transmisión de todas las ráfagas mediante la combinación de un mecanismo de asignación de caminos y longitudes de onda con esquemas básicos de resolución de contiendas. Matemáticamente, el problema de asignación de recursos en ésta estrategia es equivalente a un problema de coloreado de grafos. Así, a partir de las comunicaciones existentes en la red y de las interferencias entre los caminos utilizados por cada una de ellas, se define un grafo, sobre el que se realiza la asignación de longitudes de onda mediante el coloreado de sus vértices.

En este trabajo se han analizado, programado y evaluado cuatro algoritmos de coloreado de grafos: el *DSatur* [23] tres variantes de algoritmos *Greedy* [24], el algoritmo *VertexColoring* [28] y el *Merge*[25].

La aplicación que realiza el coloreado de grafos se ha desarrollado en Java. Incorpora 20 grafos de ejemplo que se han utilizado para comparar los distintos algoritmos seleccionados. Además, permite definir gráficamente un grafo cualquiera sobre el que realizar el coloreado. La aplicación también incorpora distintas técnicas de recoloreado (que permiten asignar más de un color a cada vértice del grafo respetando las restricciones correspondientes).

En el primer capítulo de éste documento se describen brevemente las distintas técnicas de conmutación para redes ópticas, y se explican las principales características de la tecnología de conmutación óptica de ráfagas.

En el segundo capítulo se introducen los distintos algoritmos de coloreado y las técnicas de recoloreado programadas.

El tercer capítulo describe la aplicación realizada, describiendo la interfaz gráfica y su funcionamiento.

En el cuarto capítulo se realiza una comparativa entre los distintos algoritmos de coloreado y recoloreado. Dicha comparativa se realiza en base a los resultados obtenidos utilizando los grafos de ejemplo incluidos en la aplicación.

Finalmente, se exponen las principales conclusiones del trabajo realizado.

CAPÍTULO 1. INTRODUCCIÓN A LA CONMUTACIÓN ÓPTICA DE RÁFAGAS

1.1. Paradigmas de conmutación en redes ópticas

El tráfico en Internet ha crecido exponencialmente durante los últimos años, registrando entre el 2005 y el 2007 un incremento del 60% respecto al año anterior [1]. Dicho incremento es debido a los avances tecnológicos que han permitido el desarrollo de gran diversidad de aplicaciones, desde la aparición del hipertexto en los 90 que impulsó el uso de la *web*, hasta las aplicaciones P2P de distribución, *streaming* y descarga de contenidos multimedia o los servicios de telefonía IP y videoconferencia (para citar algunos ejemplos).

Las redes ópticas permiten dar respuesta a ésta demanda creciente de ancho de banda, proporcionando retardos bajos y elevadas tasas de transmisión.

Los primeros sistemas de transmisión basados en fibras ópticas se instalaron a finales de los años 70. Se trataba de sistemas en los que sólo se utilizaba un canal por fibra y en los que la conmutación en los nodos intermedios se realizaba a nivel electrónico. A mediados de los años noventa, con la introducción de la tecnología de multiplexado por longitud de onda (*Wavelength Division Multiplexing*, WDM) que permite transmitir simultáneamente utilizando diferentes longitudes de onda dentro de una misma fibra, se consiguió incrementar la capacidad de dichas redes en varios órdenes de magnitud (alcanzando velocidades de hasta centenares de terabits por segundo [2]).

En esta evolución hacia una Internet totalmente óptica, el paradigma de conmutación destaca como un elemento clave para conseguir aprovechar de forma eficiente la capacidad disponible en dichas redes. En este sentido, la figura 1.1 muestra la evolución prevista de las redes ópticas basadas en sistemas WDM [3].

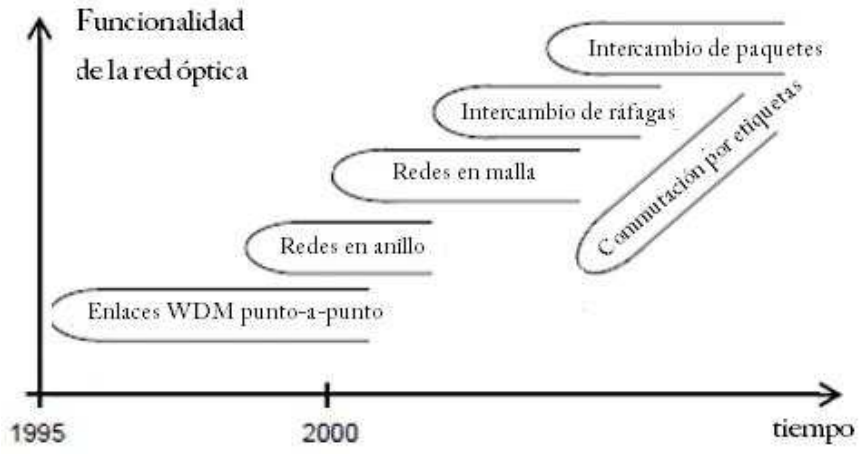

Figura 1.1 Evolución prevista de las redes basadas en WDM.

1.1.1. Conmutación óptica de circuitos

La conmutación óptica de circuitos (*Optical Circuit Switching*, OCS) consiste en el establecimiento de circuitos de luz (*lightpaths*) entre el nodo origen y el destino. Una vez establecido dicho *lightpath*, la transmisión de datos se realiza de forma transparente en el dominio óptico [4].

De forma análoga a la conmutación de circuitos en el dominio eléctrico, los recursos de la red en el paradigma OCS no se utilizan de forma óptima ya que, al estar asignados en exclusiva, se desaprovechan durante los intervalos de tiempo en los que no se realiza transmisión de datos. Además, en general, se reserva una longitud de onda para cada circuito, lo cual limita la granularidad en la asignación de los recursos de la red.

1.1.2. Conmutación óptica de paquetes

La conmutación óptica de paquetes (*Optical Packet Switching*, OPS) se basa en la transmisión de la información de los usuarios utilizando unidades de datos, paquetes, que incluyen una cabecera con la información necesaria para que los dispositivos intermedios puedan decidir hacia dónde transmitirlos [5].

La conmutación de paquetes permite el uso óptimo de los recursos de la red gracias al multiplexado estadístico de los datos. Sin embargo, existen inconvenientes particulares a dicho paradigma de conmutación en el dominio óptico. En OPS, el procesado de la cabecera de los paquetes debería realizarse directamente en el dominio óptico. No obstante, la tecnología actual no permite tal procesado a las velocidades necesarias para tomar las decisiones de encaminamiento requeridas sin necesidad de almacenar los datos de los usuarios. A éste inconveniente hay que añadir la imposibilidad de almacenar la luz debido a la falta de un equivalente real a las memorias RAM (*Random Access Memory*) en el dominio óptico. Como alternativa, existe la posibilidad de implementar retardadores utilizando fibras de retardo (*Fiber Delay Lines*, FDL). Sin embargo, dichos retardadores sólo permiten almacenar la luz durante períodos de tiempo directamente proporcionales a la longitud de la fibra utilizada. Además, se requieren bobinas de fibra de grandes dimensiones para conseguir retardos pequeños (una fibra de 1 Km de longitud sólo permite retardar la transmisión de información durante 5 μs).

1.1.3. Conmutación óptica de ráfagas

A medio camino entre OCS y OPS, se define la conmutación óptica de ráfagas (*Optical Burst Switching*, OBS). OBS se definió con la intención de aprovechar lo mejor de ambas tecnologías de conmutación. Es decir, conseguir un grado de multiplexado estadístico de usuarios superior al que se alcanza en OCS pero sin los elevados requerimientos tecnológicos que exige OPS [6].

En OBS, los datos de los usuarios se agregan en la frontera de la red formando unidades de datos denominadas ráfagas. Previamente a la transmisión de cada

ráfaga, se transmite un paquete de control con el objetivo de configurar los dispositivos intermedios para que la transmisión de la ráfaga se pueda realizar íntegramente en el dominio óptico.

En los siguientes apartados se resumen las características principales de las redes basadas en la tecnología de conmutación óptica de ráfagas.

1.2. Características básicas de las redes OBS

1.2.1. Arquitectura de la red

En una red OBS, los datos de los usuarios se agregan formando ráfagas en los nodos frontera de entrada (*ingress node*s). Antes de la transmisión de una ráfaga, el nodo frontera genera un paquete de control que es procesado electrónicamente en todos los nodos interiores de la red (*core nodes*) con el objetivo de reservar los recursos necesarios y configurar la matriz de conmutación para permitir la posterior transmisión de la ráfaga íntegramente en el dominio óptico (a través de los enlaces WDM que interconectan los distintos nodos de la red). Entre la transmisión del paquete de control y la ráfaga suele existir un intervalo de tiempo que se conoce como tiempo de *offset*. Cuando una ráfaga alcanza el nodo frontera de salida (*egress node*), se desensambla, recuperando así la información original del usuario para que siga su camino hacia el destino final [2,7].

Así pues, las principales tareas de los nodos frontera OBS son:

- Generar las ráfagas a la entrada (ensamblado) y recuperar la información original de dentro de las ráfagas a la salida (desensamblado).

- Generar el paquete de control que deberá realizar la reserva de recursos en los nodos intermedios para permitir la transmisión de la ráfaga en el dominio óptico.

- Establecer el offset entre el paquete de control y la ráfaga de datos asociada.

Mientras que las principales tareas de los nodos interiores son:

- Procesar el paquete de control y realizar la reserva de recursos pertinente. Dicha tarea requiere:

    - Programar la configuración de la matriz de conmutación.

    - Propagar la información de control hacia el siguiente nodo de la red.

- En el caso que la reserva solicitada no se pueda realizar, activar el mecanismo de resolución de contiendas pertinente.

La figura. 1.2 muestra un esquema de una red OBS.

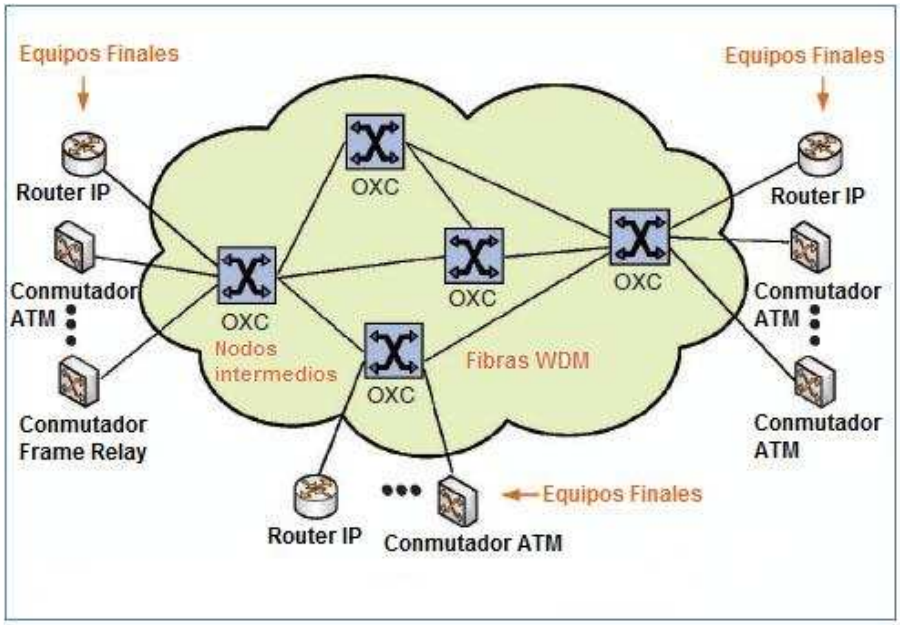

Figura 1.2 Arquitectura de una red OBS.

La figura 1.3 muestra un ejemplo de envío de una ráfaga con su respectivo paquete de control y el tiempo de offset.

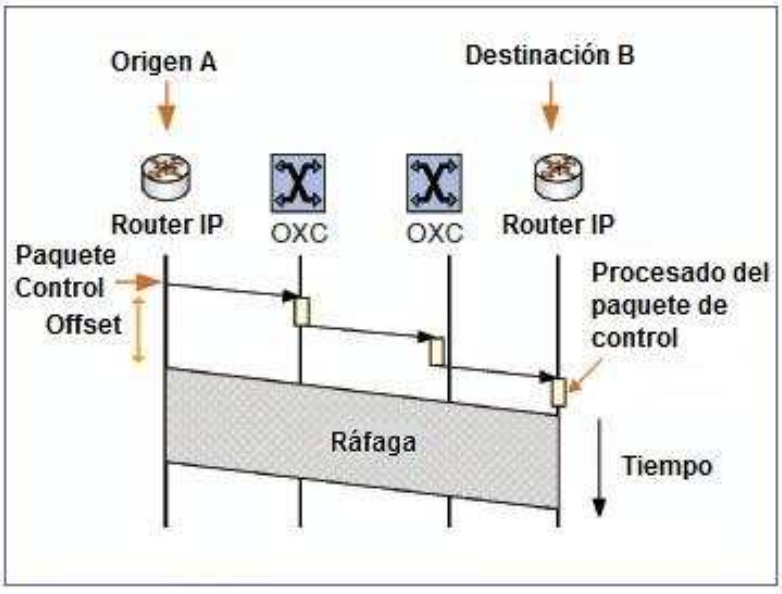

Figura 1.3 Ejemplo de envío de una ráfaga en una red OBS.

1.2.2. Mecanismos de ensamblado de ráfagas

Los principales mecanismos de generación (ensamblado) de ráfagas [8] son:

- Basados en tiempo: Tras la llegada del primer paquete se inicia un temporizador fijo de duración T. Transcurrido ese tiempo todos los paquetes que hayan llegado durante ese intervalo son enviados en una ráfaga.

- Basados en longitud: Se van almacenando los datos de los paquetes que van llegando hasta un tamaño S. Una vez alcanzado este tamaño, se envía la ráfaga.

- Híbridos: Se utilizan los dos umbrales anteriores (S y T). En el momento en el que cualquiera de ellos es alcanzado, se envía la ráfaga. La figura 1.4 muestra el diagrama de flujo de un esquema de ensamblado híbrido.

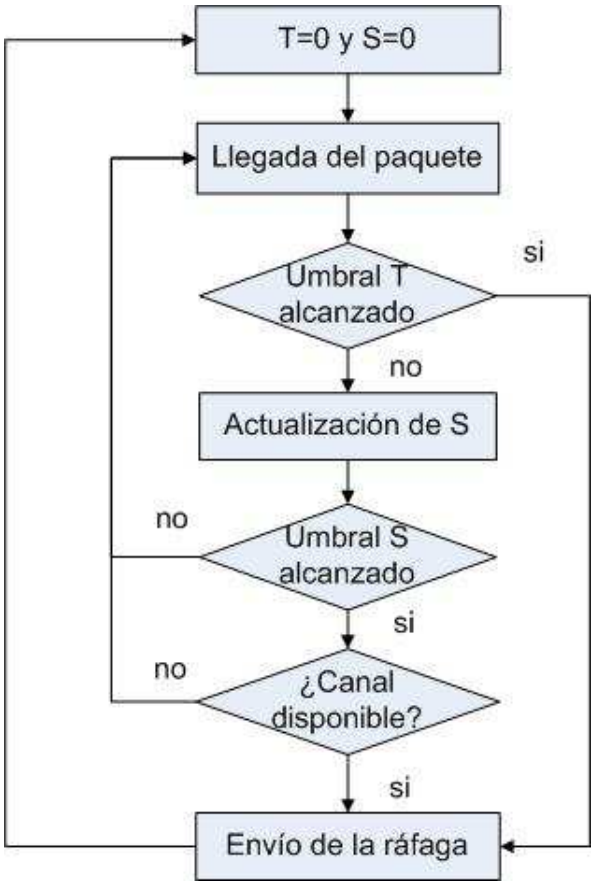

Figura 1.4 Ejemplo de mecanismo híbrido de generación de ráfagas.

El tamaño de las ráfagas es un parámetro de diseño importante que puede tener un gran impacto en las prestaciones de la red. Un tamaño de ráfaga grande puede retener los recursos durante demasiado tiempo y un tamaño muy pequeño puede conllevar una baja utilización.

1.2.3. Esquemas de señalización

La conmutación óptica de ráfagas es una adaptación de un estándar del ITU-T (*International Telecommunication Union - Telecommunication Standarization Sector*) sobre conmutación de ráfagas en ATM, denominado *ATM Block Transfer* (ABT). Los protocolos de señalización definidos en OBS derivan de las dos versiones existentes en el estándar ABT [9,10]:

- ABT con transmisión retardada: Cuando una fuente tiene una ráfaga para transmitir, primero intenta reservar el ancho de banda mediante el envío de un pequeño mensaje de solicitud. Cada nodo intermedio que recibe el mensaje de solicitud, aloja los recursos necesarios. Si el camino no se puede reservar, el nodo que no pueda realizar la reserva, envía un mensaje hacia el origen informando de lo sucedido. Transcurrido un cierto tiempo se retransmite la solicitud.

- ABT con transmisión inmediata: La fuente transmite la ráfaga sin hacer una reserva previa de recursos. En cada nodo la ráfaga es retardada mientras se configura la tabla de conmutación. Si la reserva de recursos falla en algún nodo intermedio, se envía un mensaje hacia el origen informando de lo sucedido. Transcurrido un cierto tiempo se retransmite la ráfaga.

En OBS se definen muchos esquemas de señalización, que básicamente pueden agruparse en dos grandes grupos: los basados en una reserva de recursos unidireccional (*one-way*) y los basados en reservas de recursos con reconocimiento de la disponibilidad de los mismos (*two-way*). Puesto que los esquemas unidireccionales introducen, en general, menor retardo, son los más populares y comúnmente adoptados.

Los dos principales esquemas de señalización unidireccionales definidos en OBS son:

- *Just-In-Time* (JIT) [11,12]. En el esquema JIT, la longitud de onda asignada a la transmisión de una ráfaga en un nodo se reserva desde el instante que se procesa el paquete de control hasta que se envía el último bit de la ráfaga. En el caso que la longitud de onda solicitada no pueda ser reservada, el paquete de control es rechazado y la ráfaga es descartada.

- *Just-Enough-Time* (JET) [11,12,13]. En el esquema JET, la longitud de onda asignada a la transmisión de una ráfaga en un nodo se reserva desde el instante que llega el primer bit de la ráfaga hasta que se envía el último bit de la ráfaga. De esta forma, se deja el canal disponible durante el intervalo de tiempo que transcurre entre la llegada del paquete de control y la ráfaga.

  En JET el paquete de control contiene información sobre el tiempo de llegada de la ráfaga a un nodo (que deberá actualizarse tras el retardo de procesado), y de su longitud. Esta información permite conocer

exactamente los instantes de llegada y partida de una ráfaga en un nodo intermedio de la red.

En la figura 1.5 se muestra un ejemplo de funcionamiento de ambos esquemas de señalización.

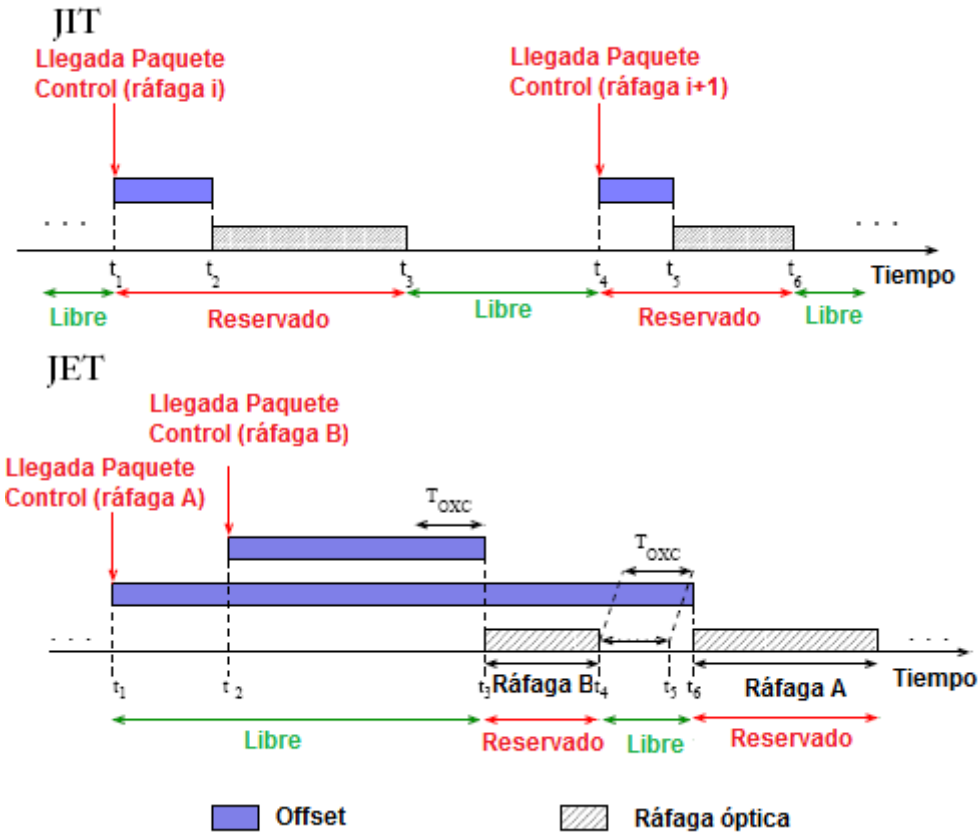

Figura 1.5 Ejemplo de funcionamiento esquema JIT y JET.

1.2.4. Mecanismos de resolución de contiendas

En los esquemas de señalización unidireccionales (*one-way*) la transmisión de la ráfaga se inicia sin que haya confirmación de la disponibilidad de los recursos solicitados en todos los nodos intermedios de la red. Así pues, es posible que dos o más ráfagas compitan simultáneamente por el uso de la misma longitud de onda en un determinado enlace de la red. Esta competencia por el uso de los recursos se conoce como contienda (*contention*).

En una red electrónica los conflictos de los paquetes por un mismo canal de salida pueden resolverse mediante almacenamiento en búferes. En OBS el uso de almacenamiento es imposible o muy limitado, por lo que se ha de abordar el problema de forma diferente [14].

Existen 3 esquemas posibles a la hora de tratar las contiendas: eliminación, desvío y preferencia [15,16]. Todos ellos pueden coexistir y operar conjuntamente.

1.2.4.1 Eliminación (dropping)

Si no se dispone de mecanismos de resolución de contiendas, cuando un paquete de control no puede reservar los recursos solicitados, la ráfaga asociada simplemente es descartada.

1.2.4.2 Desvío (deflection)

En este esquema, delante de una situación de contención, se intenta enviar la ráfaga por un canal de salida diferente al inicial mediante una de las 3 opciones siguientes:

- Nivel de longitud de onda.

  La ráfaga se envía por una longitud de onda diferente a la que, en principio, tendría que haber sido a la salida del nodo. Esto requiere de conversores de longitud de onda en los nodos que, aunque reducen considerablemente las pérdidas en una red OBS, tienen el inconveniente de ser muy caros e incrementar considerablemente el coste global de la red.

- Nivel de espacio.

  La ráfaga se envía utilizando un puerto de salida diferente al que en principio tendría que haber sido su puerto de salida predeterminado. De esta forma, la ráfaga sigue un camino alternativo para llegar a su destino.

- Nivel temporal.

  Se emula el comportamiento de los *buffers* electrónicos convencionales utilizando fibras de retardo (*Fiber Delay Lines*, FDLs) [17], retardando el envío de la ráfaga utilizando fibras auxiliares donde se almacena la ráfaga durante un tiempo determinado que depende de la longitud de la fibra. El inconveniente de estas fibras de retardo es que se necesitan Kilómetros para conseguir tiempos de retardo del orden de milisegundos.

1.2.4.3 Preferencia (preemption)

También es posible asignar prioridades a la hora de determinar qué ráfaga, de entre las que se encuentran en situación de contención, puede reservar los

recursos solicitados. De esta forma, se permite deshacer una reserva previamente realizada para ceder los recursos a una ráfaga de mayor prioridad.

1.2.5. Optimización en redes OBS

Además de los mecanismos básicos de resolución de contiendas explicados en el apartado anterior, existen varias propuestas que combinan algunas de las técnicas de resolución de contenciones básicas para minimizar las pérdidas en redes OBS [18].

En este sentido, en [19] los autores proponen una estrategia de transmisión que garantiza pérdidas nulas en redes OBS mediante la combinación de un esquema de asignación de caminos y longitudes de onda con el uso de esquemas básicos de resolución de contiendas. En esta estrategia, se modelan las comunicaciones y el nivel de interferencia entre ellas mediante un grafo, llamado digrafo de restricciones.

Como ejemplo, suponer que se desea definir el digrafo de restricciones considerando la red representada en la figura 1.6 en la cual se definen tres comunicaciones (denotadas como $p_{1,3}$, $p_{2,3}$ y $p_{4,3}$).

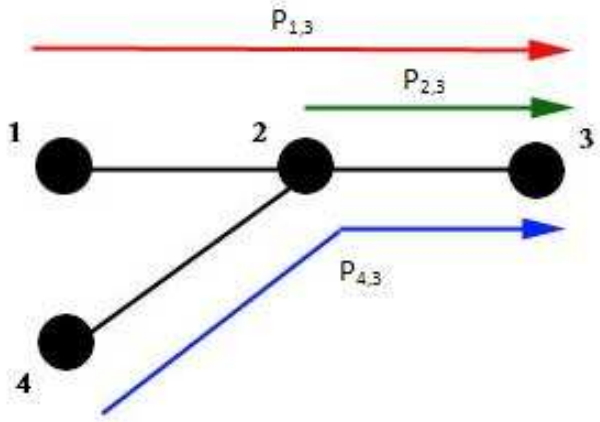

Figura 1.6 Ejemplo de transmisión red OBS.

Así pues, el digrafo de restricciones correspondiente tiene tres vértices (tal y como muestra la figura 1.7) y los arcos y aristas que los unen se definen en función de las interferencias entre las tres comunicaciones, tal y como se explica a continuación.

El tipo de interferencia que existe entre las comunicaciones $p_{1,3}$ y $p_{4,3}$ se denomina colisión (*collision*) puesto que los caminos utilizados por éstas comunicaciones comparten un enlace intermedio (el 2-3) siendo el enlace que le precede distinto en cada uno de los caminos (siendo el 1-2 para la comunicación $p_{1,3}$ y el 4-2, para la $p_{4,3}$). Las colisiones se representan mediante

aristas. Así, en el digrafo de restricciones se dibuja una arista entre los vértices $p_{1,3}$ y $p_{4,3}$.

El tipo de interferencia que existe entre las comunicaciones $p_{1,3}$ y $p_{2,3}$ se denomina solapamiento (*overlapping*). En particular, se dice que $p_{1,3}$ solapa $p_{2,3}$ porque ambas comunicaciones no colisionan y el primer enlace de la comunicación $p_{2,3}$ corresponde a un enlace intermedio del camino utilizado por la comunicación $p_{1,3}$. Los solapamientos se representan mediante arcos. Así, en el digrafo de restricciones se dibuja un arco con origen en el vértice $p_{1,3}$ y destino en el vértice $p_{2,3}$. Lo mismo sucede para las comunicaciones $p_{4,3}$ y $p_{2,3}$.

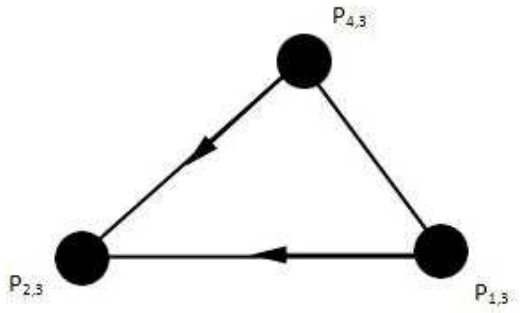

Figura 1.7 Digrafo de restricciones correspondiente a la red de la figura 1.6

Para conseguir pérdidas nulas, el esquema de asignación de caminos y longitudes de onda propuesto en esta estrategia debe combinarse con uno de los dos mecanismos de resolución de contiendas siguientes:

- Una única FDL en cada nodo intermedio.

- Un tiempo de *offset* adecuado para cada comunicación.

En el primer caso, utilizando una FDL en cada nodo intermedio, las interferencias por solapamiento no suponen una restricción en la asignación de longitudes de onda, de modo que el digrafo de restricciones a considerar para realizar dicha asignación se puede simplificar considerando simplemente las aristas del digrafo (y eliminando los arcos).

La figura 1.8 muestra el grafo obtenido tras eliminar los arcos del digrafo de restricciones correspondiente al esquema de transmisión de la figura 1.6.

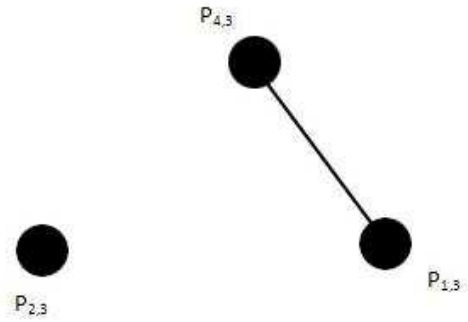

Figura 1.8 Grafo obtenido a partir del esquema de transmisión de la figura 1.6 sobre el cual hay que realizar el coloreado.

En el segundo caso, utilizando el esquema de resolución de contiendas mediante la selección del tiempo de offset de cada comunicación, hay que garantizar que, cuando existe un ciclo (dirigido) en el digrafo de restricciones, al menos una de las comunicaciones del ciclo recibe una longitud de onda distinta a las demás comunicaciones del ciclo. Así, para la asignación de longitudes se puede simplificar el digrafo de restricciones eliminando todos los arcos pero dejando al menos una arista por ciclo.

En cualquier caso, el problema de asignación de longitudes de onda propuesto en este esquema sin pérdidas se puede modelar matemáticamente como un problema de coloreado de grafos, utilizando el grafo resultante de simplificar el digrafo de restricciones.

En el capítulo siguiente se describen el conjunto de algoritmos de coloreado de grafos que se han considerado, programado y evaluado en este trabajo.

CAPÍTULO 2. ASIGNACIÓN DE LONGITUDES DE ONDA MEDIANTE TÉCNICAS DE COLOREADO DE GRAFOS

2.1. Definición de grafo

Un grafo es un conjunto de vértices o nodos conectados por aristas o arcos [20]. Así pues, un grafo *G* con *n* vértices consiste en un conjunto de vértices *V*, tal que $|V| = n$, y un conjunto de aristas *E*, donde cada arista es un par desordenado de vértices. A lo largo de este capítulo los vértices de *G* se denotarán con enteros 1, 2, …, *n*. Si el par desordenado de vértices *{u, v}* están unidos por una arista de *G*, entonces los vértices *u* y *v* son vecinos, o adyacentes, y se denota como $uv \in E$. El hecho de que dos nodos sean vecinos, es una relación simétrica, de forma que: $uv \in E$ si y solo si $vu \in E$.

2.2. Asignación de longitudes de onda mediante el coloreado de grafos

El uso de algoritmos de coloreado de grafos para realizar la asignación de longitudes de onda en redes de comunicaciones es una técnica muy utilizada, sobretodo en conmutación de circuitos [21], pero también puede aplicarse a la asignación de recursos en redes OBS.

El coloreado de un grafo consiste en asignar colores a sus vértices de tal modo que vértices adyacentes reciban colores distintos y, en la medida de lo posible, minimizando el número total de colores utilizados. En redes ópticas, en general, cada color equivale a una longitud de onda. Para el caso particular de redes OBS, antes de realizar el coloreado es necesario definir un grafo que refleje las restricciones que se consideren oportunas. En [22] los autores proponen la definición del denominado Digrafo de Restricciones que refleja el nivel de interferencia entre comunicaciones y sobre el cual se pueden aplicar algoritmos de coloreado para realizar la asignación de longitudes de onda a las distintas comunicaciones.

A continuación, se describen el conjunto de algoritmos de coloreado de grafos que se han estudiado, programado y evaluado en este proyecto.

2.3. Algoritmos de coloreados de grafos

2.3.1. Algoritmo DSatur

El algoritmo *DSatur* fue implementado por Daniel Brélaz en el año 1979 [23].

Este algoritmo es un método heurístico de coloreado de grafos basado en el grado de saturación de un vértice; donde el grado de saturación es el número de colores diferentes a los cuales este vértice es adyacente.

Los pasos que sigue el algoritmo son los siguientes:

1. Ordenar los vértices de forma decreciente según su grado (definido como el número total de vértices vecinos).

2. Colorear el vértice que presente el mayor grado con el color 1. En caso de igualdad elegir un vértice cualquiera con el mismo grado.

3. Seleccionar el vértice con el máximo grado de saturación. Si existe igualdad, seleccionar uno de los vértices con mayor grado que aún no haya sido coloreado.

4. Colorear el vértice seleccionado con el mínimo color posible (cada color corresponde a un número entero, de modo que se ha de elegir el color que equivale al menor entero posible).

5. Repetir los pasos 3 y 4 tantas veces como sea necesario hasta que todos los vértices del grafo estén coloreados.

El pseudocódigo del algoritmo *DSatur* sería el siguiente:

```
Requiere: La matriz de adyacencias del grafo y un vector
con los colores que se van asignando.

Resultado: Coloreado del grafo dado con el algoritmo
DSatur.

while queden vértices por colorear do

     for cada vértice que no haya sido coloreado do
          Calcular su grado de saturación.
     end for

     for cada vértice que no haya sido coloreado do
          if su grado de saturación es mayor do
               Almacenar este valor.
          end if
     end for

     for cada vértice no coloreado do
          Ordenar estos vértices de mayor a menor grado
          de saturación.
     end for

     Asignar color al primer vértice de la lista.

end while
```

En la figura 2.1 se muestra un diagrama con los pasos lógicos que realiza el algoritmo *DSatur*.

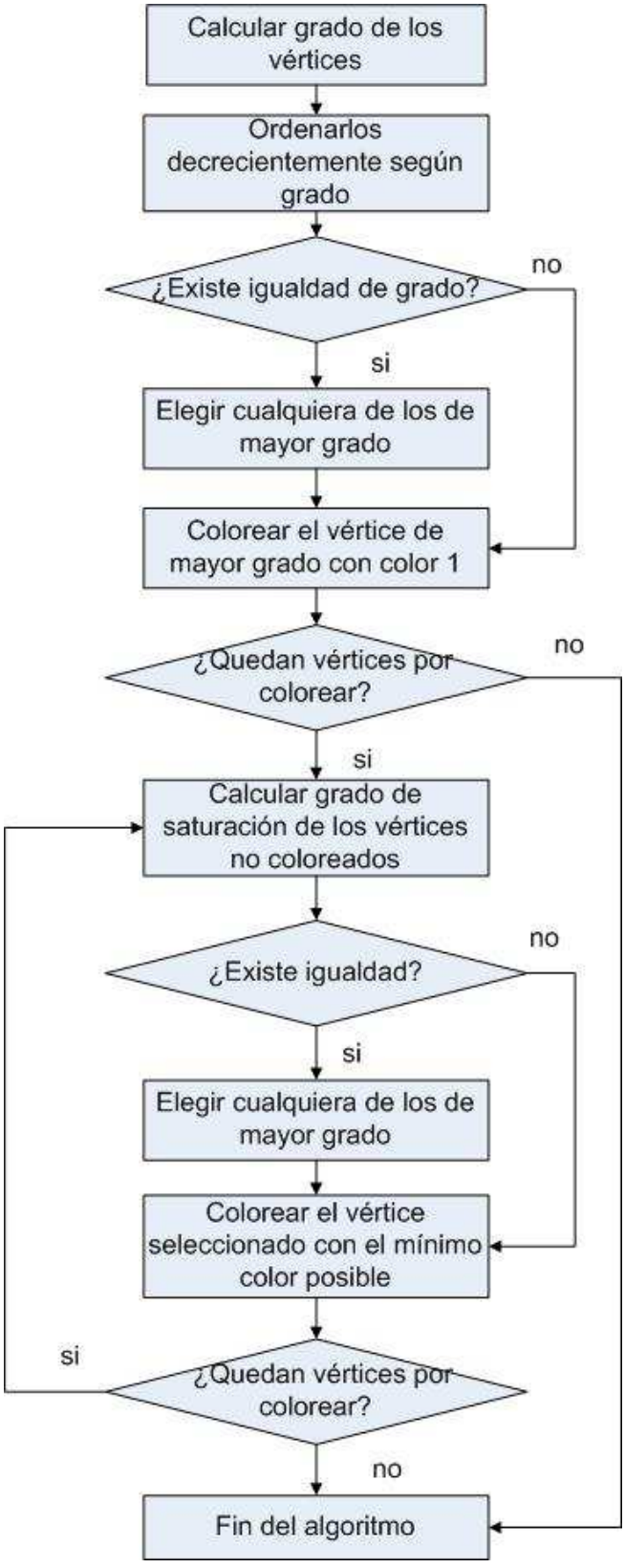

Figura 2.1 Diagrama *DSatur*.

A continuación se explica un ejemplo para ilustrar el funcionamiento del algoritmo. Suponer que se desea colorear el grafo de la figura 2.2. En primer lugar, hay que ordenar los vértices según su grado, en orden decreciente (tal y como muestra la tabla 2.1).

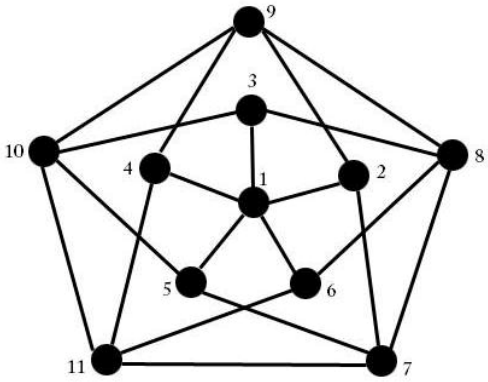

Figura 2.2 Ejemplo de grafo a colorear con el algoritmo *DSatur*.

Tabla 2.1 Grado de los vértices del grafo de la figura 2.2 y su ordenado.

Vértice	Grado (nº conexiones)
1	5
7	4
8	4
9	4
10	4
11	4
2	3
3	3
4	3
5	3
6	3

Debido a que solamente el vértice 1 tiene grado 5 y no hay ningún otro vértice con un grado superior a él. El vértice 1 es coloreado con el color 1.

Una vez coloreado el primer vértice, se ordenan los vértices de manera decreciente según su grado de saturación (como muestra la tabla 2.2). Hay que tener en cuenta que se ha descartado ya el vértice 1 porque ya tiene un color asignado.

Tabla 2.2 Grado de saturación de los vértices del grafo de la figura 2.2.

Vértice	Grado	Grado de saturación
2	3	1
3	3	1
4	3	1
5	3	1
6	3	1
7	4	0
8	4	0
9	4	0
10	4	0
11	4	0

Debido a que los vértices del 2 al 6 tienen el mismo grado, se elige cualquiera de ellos. En la presente implementación del algoritmo, en caso de igualdad, se elige el vértice con un identificador menor. Así pues, el siguiente vértice elegido para asignarle un color es el vértice 2 y como es adyacente al vértice 1, con color 1, se le asigna el color 2.

```
Requiere: La matriz de adyacencias del grafo, la matriz
que almacena el grado y el grado de saturación de cada
vértice.

Resultado: Vector con los vértices ordenados.

for cada uno de los vértices do
      if dos vértices tienen el mismo grado && el mismo
      grado de saturación do
        if el primer vértice tiene mayor identificador do
           Intercambiar el vértice con menor
           identificador a una posición preferente.
        end if
      end if
end for
```

Nuevamente se ordenan los vértices aún no coloreados en función del grado de saturación, tal y como muestra la tabla 2.3.

De la lista a continuación, se obtiene el próximo vértice a colorear, el 7 ya que tiene mayor grado que cualquiera de los vértices del 3 al 6 y menor identificador que el 9. Debido a que de los dos vértices coloreados anteriormente, sólo está conectado al vértice 2, puede tener el mismo color que el vértice 1. Por lo tanto el vértice 7 es coloreado con el color 1.

Tabla 2.3 Grado de saturación de los vértices del grafo de la figura 2.2.

Vértice	Grado	Grado de saturación
7	4	1
9	4	1
3	3	1
4	3	1
5	3	1
6	3	1
8	4	0
10	4	0
11	4	0

Nuevamente se realiza la ordenación de los vértices aún no coloreados según su grado y su grado de saturación (tal y como muestra la tabla 2.4).

Tabla 2.4 Grado de saturación de los vértices del grafo de la figura 2.2.

Vértice	Grado	Grado de saturación
8	4	1
9	4	1
11	4	1
3	3	1
4	3	1
5	3	1
6	3	1
10	4	0

De la lista anterior, se deduce que el siguiente vértice a colorear es el 8. Debido a que de los vértices coloreados hasta el momento, sólo está conectado al vértice 7 coloreado con el color 1, puede ser coloreado con el color 2.

Se repiten los pasos anteriores hasta asignar un color a cada vértice. La figura 2.3 muestra el resultado final del coloreado.

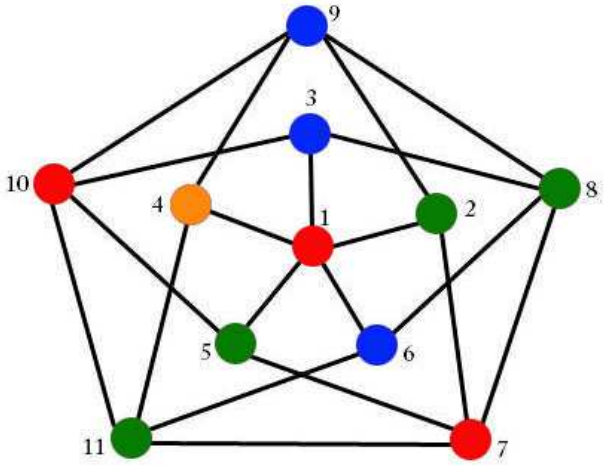

Figura 2.3 Colores asignados a los vértices del grafo de la figura 2.2 después de aplicarle el algoritmo *DSatur*.

2.3.2. Algoritmos Greedy

Los algoritmos *Greedy* (voraces) son aquellos que, para resolver un problema, siguen una metaheurística consistente en elegir una opción óptima en cada paso local con la esperanza de llegar a una solución global óptima [24]. En general, son algoritmos rápidos y fáciles de implementar, aunque no siempre aseguran alcanzar la solución óptima.

Aplicados al coloreado de grafos, las 3 variantes de algoritmos *Greedy* seleccionadas definen el criterio para elegir el vértice a colorear al inicio del algoritmo y este criterio no varía a lo largo de la resolución.

```
Requiere: La matriz de adyacencias modificada del grafo
(donde un -1 en la matriz indica que hay adyacencia entre
dos vértices).

Resultado: Coloreado del grafo dado con el algoritmo
Greedy.

Realizar la ordenación de los vértices según el criterio
de cada técnica.

for cada uno de los vértices do
    Buscar los colores a los que está conectado (números
    superiores a 0 en la fila).
    Colorear con el menor color posible disponible.
    Añadir este valor a la matriz de adyacencias.
end for
```

2.3.2.1 Algoritmo Greedy First Fit

La técnica *First Fit* es la más sencilla de todos los algoritmos *Greedy*. Este algoritmo consiste en ordenar secuencialmente los vértices según su identificador y asignar a cada uno el mínimo color posible.

```
Requiere: La matriz de adyacencias modificada del grafo
(donde un -1 en la matriz indica que hay adyacencia entre
dos vértices).

Resultado: Vector de vértices ordenados según la técnica
Greedy First Fit.

for cada uno de los vértices do
    Ordenar los vértices crecientemente según su
    identificador.
end for
```

A continuación se explica un ejemplo para ilustrar el funcionamiento del algoritmo.

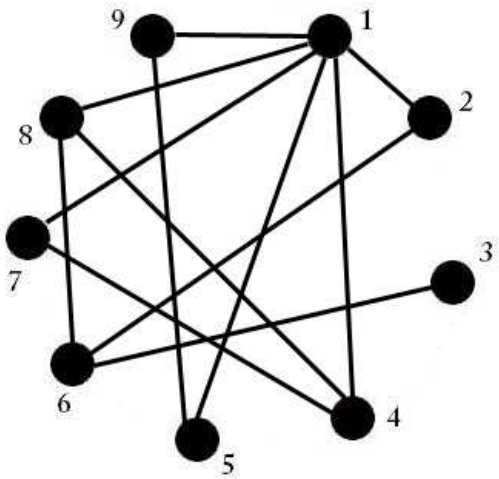

Figura 2.4 Ejemplo de grafo para colorear con el algoritmo *Greedy*.

Suponer que se quiere colorear el grafo de la figura 2.4, cuya matriz de adyacencias es la que se muestra en la tabla 2.5 (donde cada -1 indica que hay adyacencia entre dos vértices):

Tabla 2.5 Matriz de adyacencias (ya modificada) del grafo de la figura 2.4.

	1	2	3	4	5	6	7	8	9
1		-1		-1	-1		-1	-1	-1
2	-1					-1			
3						-1			
4	-1						-1	-1	
5	-1								-1
6		-1	-1					-1	
7	-1			-1					
8	-1			-1		-1			
9	-1				-1				

Inicialmente se asigna el color 1 al vértice 1 y se sustituyen todos los -1 de la primera columna de la matriz de adyacencias (ya modificada) por 1s (tal y como muestra la tabla 2.6).

Tabla 2.6 Matriz de adyacencias (ya modificada) del grafo de la figura 2.4 (primera iteración)

	1	2	3	4	5	6	7	8	9
1		-1		-1	-1		-1	-1	-1
2	1					-1			
3						-1			
4	1						-1	-1	
5	1								-1
6		-1	-1					-1	
7	1			-1					
8	1			-1		-1			
9	1				-1				

A continuación hay que colorear el vértice 2. Debido a que es adyacente al vértice 1 no le podemos asignar el mismo color que él (hay un 1 en la fila del vértice 2). De modo que el vértice 2 es coloreado con un nuevo color, que es el color 2.

Se modifica la matriz de adyacencias sustituyendo los -1 de la segunda columna de la matriz por 2s (tal y como muestra la tabla 2.7).

Tabla 2.7 Matriz de adyacencias (ya modificada) del grafo de la figura 2.4 (segunda iteración)

	1	2	3	4	5	6	7	8	9
1		2		-1	-1		-1	-1	-1
2	1					-1			
3						-1			
4	1						-1	-1	
5	1								-1
6		2	-1					-1	
7	1			-1					
8	1			-1		-1			
9	1				-1				

Se repite el procedimiento hasta que todos los vértices del grafo tengan un color asignado. La figura 2.5 muestra el resultado final del coloreado.

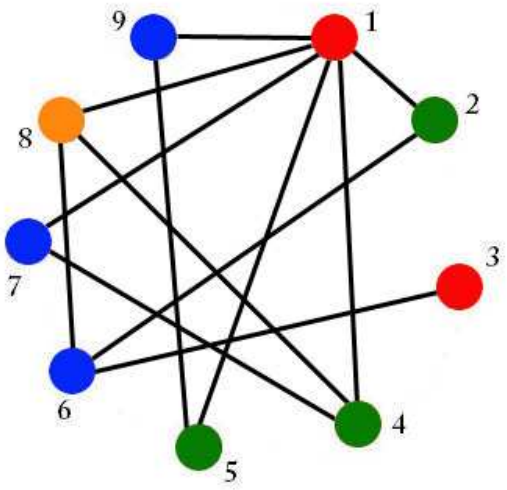

Figura 2.5 Colores asignados a los vértices del grafo de la figura 2.4 después de aplicarle el algoritmo *Greedy First Fit*.

2.3.2.2 Algoritmo Greedy Degree Based Ordering

Los algoritmos *Degree Based Ordering* se basan en ordenar los vértices en función de su grado (número total de vecinos) antes de realizar el coloreado. La ordenación se puede realizar en orden decreciente de grado (*Largest Degree Ordering*) o en orden creciente de grado (*Minimum Degree Ordering*).

Largest Degree Ordering (*LDO*)

El pseudocódigo de esta técnica es muy similar al explicado anteriormente para la técnica *First Fist* pero con una reordenación previa de los vértices en función de su grado.

```
Requiere: La matriz de adyacencias modificada del grafo
(donde un -1 en la matriz indica que hay adyacencia entre
dos vértices).

Resultado: Vector de vértices ordenados según la técnica
Greedy LDO.

for cada uno de los vértices do
    if grado es menor que el siguiente do
       Intercambiar posiciones dando preferencia al de
       mayor grado.
    end if
end for
```

Como ejemplo, suponer que se desea colorear el grafo de la figura 2.4. Inicialmente, se ordenan los vértices en orden decreciente de grado, tal y como muestra la tabla 2.8.

Tabla 2.8 Grado de los vértices del grafo de la figura 2.4 ordenados decrecientemente.

Vértice	Grado (nº conexiones)
1	6
4	3
6	3
8	3
2	2
5	2
7	2
9	2
3	1

El primer vértice a colorear es el vértice 1, al que se le asigna el color 1.

El siguiente vértice es el 4, que al ser adyacente al vértice 1 debe recibir un nuevo color, el color 2. El siguiente vértice es el 6, al que se asigna el color 1 porque no es adyacente al vértice 1. El vértice 8 es adyacente al vértice 1 (con color 1) y al vértice 4 (con color 2). Por eso se le asigna el color 3.

La figura 2.6 muestra el resultado final del coloreado, después de repetir el mismo proceso hasta que todos los vértices tengan asignado un color.

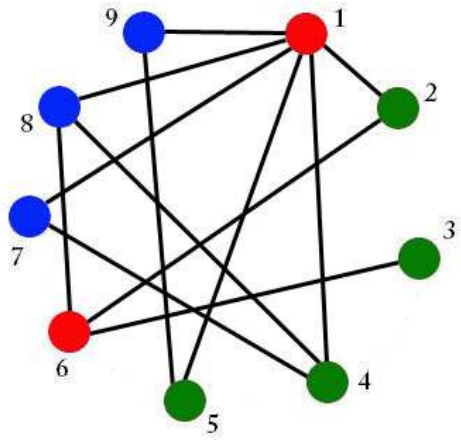

Figura 2.6 Colores asignados a los vértices del grafo de la figura 2.4 después de aplicarle el algoritmo Greedy LDO.

En este ejemplo, sólo son necesarios 3 colores (uno menos de los requeridos utilizando el algoritmo *First Fist* explicado anteriormente).

Minimum Degree Ordering (*MDO*)

El pseudocódigo es similar a la técnica del *LDO* pero ordenando los vértices de forma creciente.

```
Requiere: La matriz de adyacencias modificada del grafo
(donde un -1 en la matriz indica que hay adyacencia entre
dos vértices).

Resultado: Vector de vértices ordenados según la técnica
Greedy MDO.

for cada uno de los vértices do
    if grado es mayor que el siguiente do
       Intercambiar posiciones dando preferencia al de
       menor grado.
    end if
end for
```

Utilizando nuevamente el grafo de la figura 2.4 como ejemplo, la tabla 2.9 muestra el orden de los vértices para realizar el coloreado según el algoritmo *MDO*.

Tabla 2.9 Vértices del grafo de la figura 2.4 ordenados en orden creciente de grado.

Vértice	Grado (nº conexiones)
3	1
2	2
5	2
7	2
9	2
4	3
6	3
8	3
1	6

En este caso, el resultado final del coloreado, para el cual se requieren 3 colores, es el mostrado en la figura 2.7.

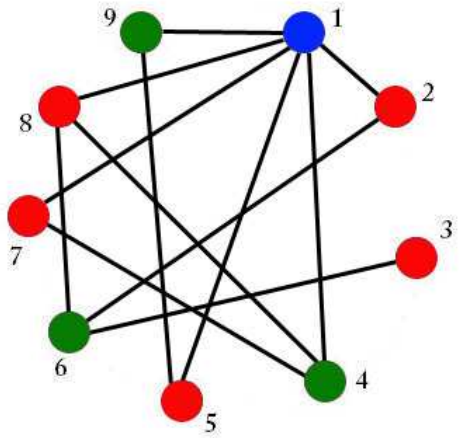

Figura 2.7 Colores asignados a los vértices del grafo de la figura 2.4 después de aplicarle el algoritmo *Greedy MDO*.

A modo de resumen, la figura 2.8 muestra un diagrama del funcionamiento de las tres técnicas *Greedy* comentadas anteriormente.

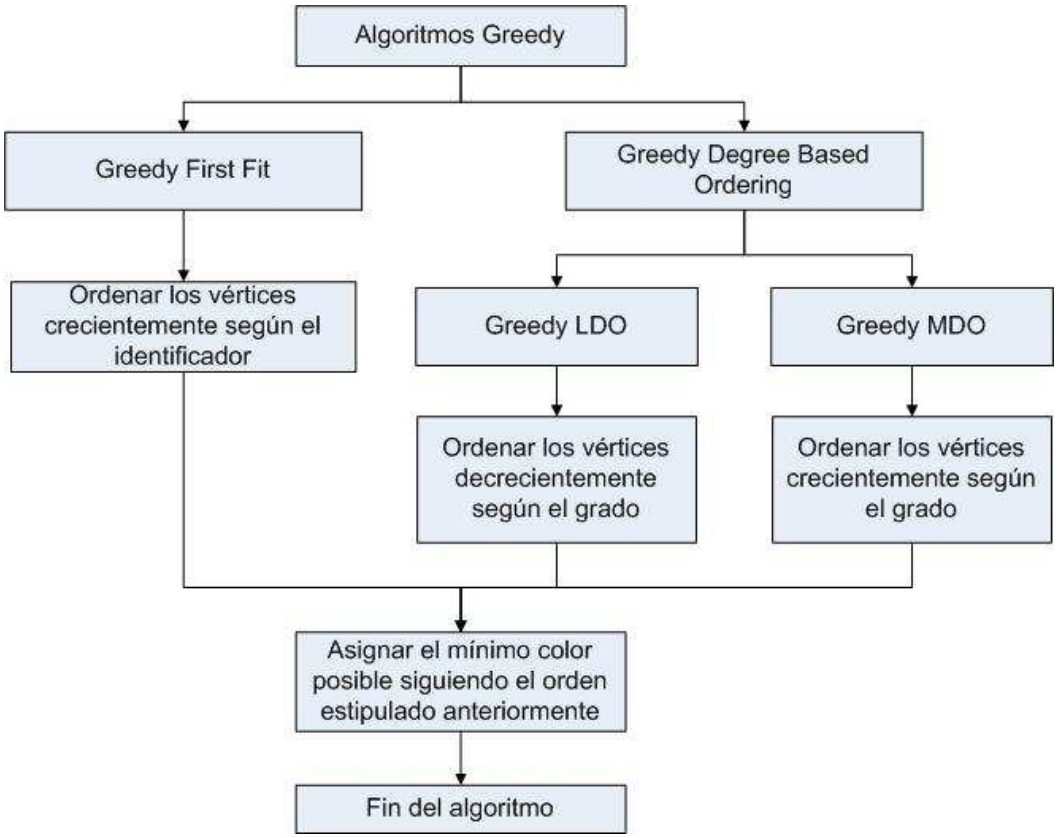

Figura 2.8 Diagrama *Greedy*.

2.3.3. Algoritmo Merge

El algoritmo *Merge* de R.D. Dutton y R.C. Brighamt [25], para el coloreado de grafos conexos, está basado en el algoritmo Brualdi [26], inspirado por el teorema de ZyKov [27].

Este algoritmo realiza la unión de los vértices del grafo hasta obtener un grafo completo en el mayor número de pasos posible. Los pasos que sigue el algoritmo son los siguientes:

1. Organizar los vértices no adyacentes a pares según el número de vértices adyacentes comunes.

2. Unir el par de vértices con un mayor número de vértices adyacentes comunes. En caso de igualdad, seleccionar cualquiera de ellos.

3. Repetir los 2 pasos anteriores hasta que todos los vértices sean adyacentes.

4. Asignar el mismo color a todos los vértices que son representados por un mismo vértice en el grafo final.

El pseudocódigo implementado a continuación junto con el diagrama de la figura 2.9 explican el funcionamiento de este algoritmo.

```
Requiere: La matriz de adyacencias del grafo.

Resultado: Coloreado del grafo dado con el algoritmo Merge.

for cada uno de los vértices do
    if no están conectados (grafo[x][y]==0) do
       Contar conexiones (1s) en común y
       registrarlo en una matriz "conexiones".
    end if
end for

for cada uno de los vértices do
    Encontrar el mayor número dentro de la matriz
    "conexiones".
end for

for cada uno de los vértices do
    Juntar ambos vértices.
    Agrupar sus conexiones.
    Registrar su unión en una matriz (MatrizMerge) que
    almacena los grupos formados.
end for

for cada uno de los vértices do
    Análisis de la matriz MatrizMerge y asignación del
    mismo color a los vértices de un mismo grupo.
end for
```

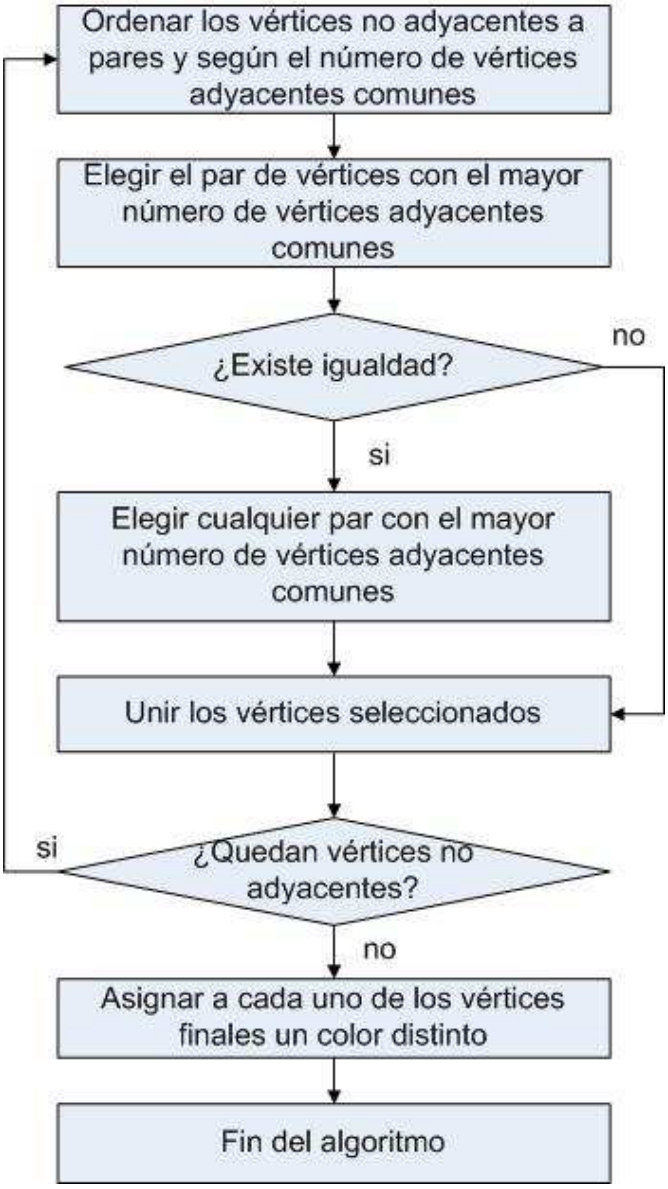

Figura 2.9 Diagrama *Merge*.

Como ejemplo, suponer que se desea colorear el grafo de la figura 2.10.

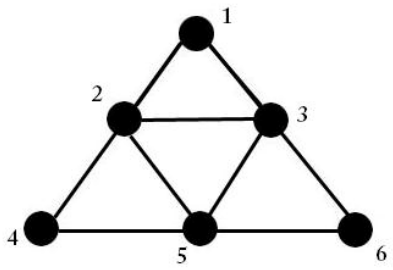

Figura 2.10 Ejemplo de grafo para colorear con el algoritmo *Merge*.

Primero, se identifican todos los pares de vértices no adyacentes y se calcula el número de vértices adyacentes de cada par (tal y como muestra la tabla 2.10).

Tabla 2.10 Pares de vértices no adyacentes de la figura 2.10 y vértices adyacentes comunes a cada par.

Pares de vértices	Vértices adyacentes comunes
1, 4	1
1, 5	2
1, 6	1
2, 6	2
3, 4	2
4, 6	1

Ahora hay que juntar los dos vértices de aquél par que tengan un mayor número de vértices adyacentes comunes. En este caso, se elige el par 3,4 (aunque también sería posible elegir el par 1,5 o 2,6).

Después de unir los vértices 3 y 4, el grafo resultante es el que se muestra en la figura 2.11.

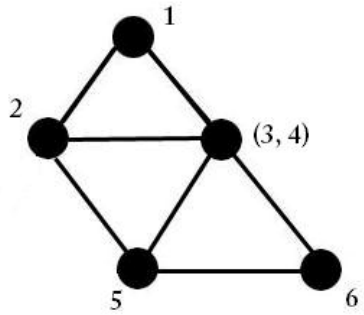

Figura 2.11 Grafo resultante tras la unión de los vértices 3 y 4.

Sobre el grafo resultante, se vuelven a identificar los pares de vértices no adyacentes y el número de vértices adyacentes comunes. El resultado se muestra en la tabla 2.11.

Tabla 2.11 Pares de vértices no adyacentes de la figura 2.11 y vértices adyacentes comunes a cada par.

Pares de vértices	Vértices adyacentes comunes
1, 5	2
1, 6	2
2, 6	2

Se seleccionan los vértices 2 y 6 (aunque cualquiera de los tres pares podría ser el seleccionado). El resultado es el que se muestra en la figura 2.12.

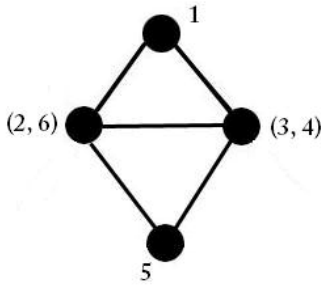

Figura 2.12 Grafo resultante tras la unión de los vértices 2 y 6.

Los únicos vértices no adyacentes que quedan por unir son los vértices 1 y 5. Una vez unidos el grafo resultante es el representado en la figura 2.13.

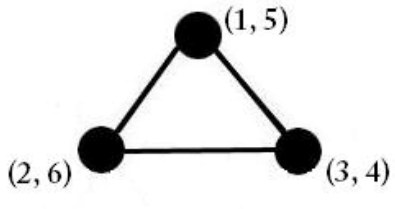

Figura 2.13 Grafo resultante tras la unión de los vértices 1 y 5.

Finalmente, sólo queda asignar el mismo color a todos los vértices que son representados por un mismo vértice en el grafo final (figura 2.13). Así pues, el resultado del coloreado es el que muestra la figura 2.14.

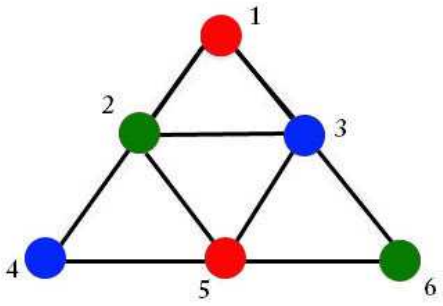

Figura 2.14 Colores asignados a los vértices del grafo de la figura 2.10 después de aplicarle el algoritmo *Merge*.

2.3.4. Algoritmo Vertex Coloring

El algoritmo *Vertex Coloring* fue implementado por Ashay Dharwadker en el año 2006 [28].

Para entender el funcionamiento del algoritmo es necesario introducir algunos conceptos previos.

El producto cartesiano de dos grafos *G* y *H*, denotado $G \times H$, es el grafo cuyo conjunto de vértices son $V(G) \times V(H)$ y cuyas aristas conectan los vértices (u_1,v_1) con los vértices (u_2,v_2) si y sólo si $u_1=u_2$ y $\{v_1,v_2\}$ es una arista *de H* o $v_1=v_2$ y $\{u_1,u_2\}$ es una arista *de G*.

Un grafo completo con *m* vértices es denominado como K_m, *donde m* es el número de colores necesarios para el coloreado.

Un conjunto independiente *S* de un grafo *G* es un conjunto de vértices de *G* no adyacentes entre sí. Dado un conjunto independiente *S* de *G* y un vértice *v* no perteneciente a *S*, se dice que *v* se puede añadir al conjunto, si el conjunto $S \cup \{v\}$ sigue siendo un conjunto independiente de *G*. Se denota $\rho(S)$ al número de vértices de *G* que se pueden añadir al conjunto independiente *S*. Un conjunto independiente máximo es un conjunto independiente al cuál no se le puede añadir ningún otro vértice.

El número cromático $\chi(G)$ de un grafo *G* es el menor valor de *m* para el cual existe un *m*-coloreado de los vértices de *G* [28].

Dado un grafo *G,* y suponiendo que se desea colorear el grafo con *m* colores, los pasos a realizar por el algoritmo son los siguientes:

1. Construir el producto Cartesiano $G \times K_m$

2. Sobre el grafo resultante del paso 1, buscar un conjunto independiente *S*.

3. Buscar el conjunto de vértices que se pueden unir a *S* sin que *S* deje de ser un conjunto independiente.

4. Seleccionar el vértice para el cual se maximiza $\rho(S \cup \{(u, v)\})$. Si existe igualdad elegir al azar uno de ellos.

5. Repetir los pasos 3 y 4 hasta que no queden vértices a añadir, es decir hasta que $\rho(S)$ sea 0.

6. Finalmente, analizar cada vértice del conjunto independiente para determinar el color de cada uno de los vértices del grafo original de la siguiente manera: sea $\{u,v\}$ un vértice del conjunto independiente, entonces al vértice *u* del grafo original se le debe asignar el color *v*.

A continuación se muestra un diagrama del algoritmo *Vertex Coloring.*

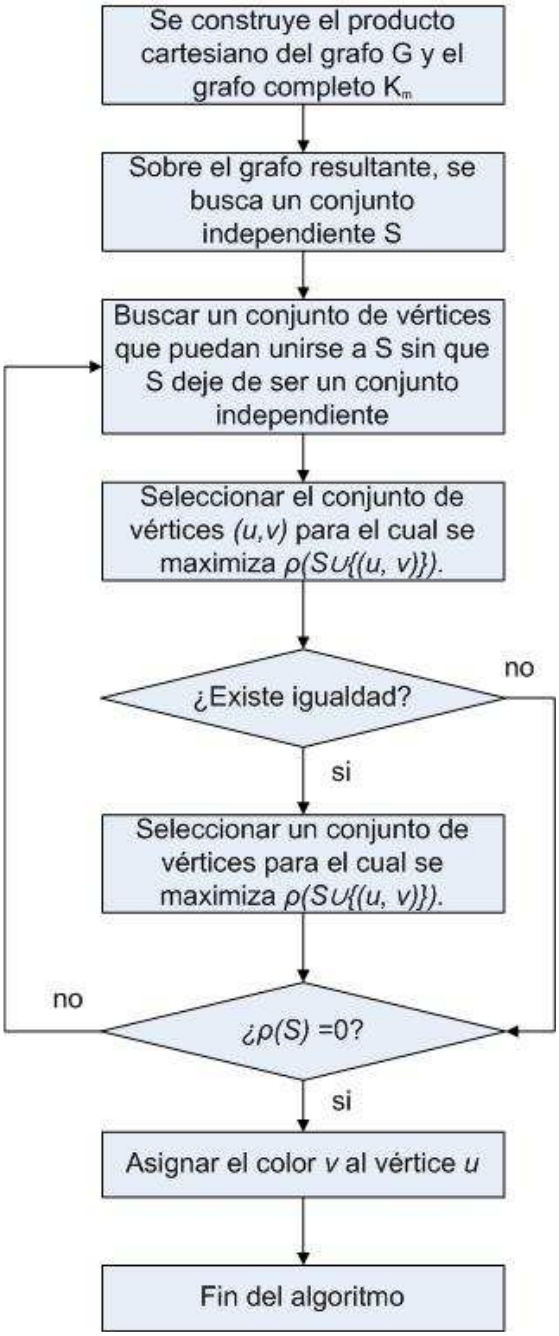

Figura 2.15 Diagrama *Vertex Coloring*.

Notar que, a diferencia de los algoritmos explicados anteriormente, en este caso es necesario conocer el número de colores con el que se realizará el coloreado antes de ejecutar el algoritmo.

A continuación se presentan partes del pseudocódigo por separado donde se muestran los pasos.

Requiere: El número de colores para realizar el coloreado.

Resultado: Se crea el grafo completo para realizar el producto cartesiano.

Ejemplo de grafo completo de 4 colores.

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Se declara una matriz de dimensiones “NumeroDeColores x NumeroDeColores”.

for cada uno de los colores **do**
    for cada uno de los colores **do**
        Se crea una matriz completa de ‘1s’.
    end for
end for

for para cada uno de los colores **do**
    Se asigna a la diagonal ‘0s’ ya que no hay conexiones entre un mismo vértice.
end for

Se realiza la multiplicación cartesiana y se procede a la búsqueda de los conjuntos independientes.

Requiere: El grafo resultante de la multiplicación cartesiana.

Resultado: El conjunto independiente final.

Seleccionar el primer vértice como primer conjunto independiente.

While se puedan añadir más vértices al conjunto independiente **do**
    Buscar los vértices no conectados al conjunto independiente y almacenarlos en un vector.

    for cada uno de los vértices no conectados **do**
        Realizar parejas del conjunto independiente más uno de los vértices no conectados.

```
        for cada uno de los vértices no conectados do
         Comprobar la cantidad de vértices que se
         le pueden unir.
         Almacenar cual registra el mayor valor.
        end for
    end for

    Crear un nuevo conjunto independiente con la pareja
    de, conjunto independiente más vértice, que hayan
    mostrado el mayor valor.
end while
```

Como ejemplo del funcionamiento del algoritmo, suponer que se desea colorear el grafo que se muestra a la izquierda de la figura. 2.16 utilizando 3 colores. Para ello, es necesario realizar el producto cartesiano de dicho grafo con el grafo completo de 3 vértices mostrado a la derecha de la figura 2.16.

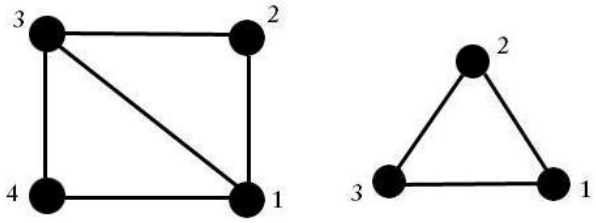

Figura 2.16 Ejemplo de grafo para realizar el coloreado con el algoritmo Vertex Coloring junto con el grafo completo K_m.

El resultado del producto cartesiano, GxK_3, se muestra en la figura 2.17.

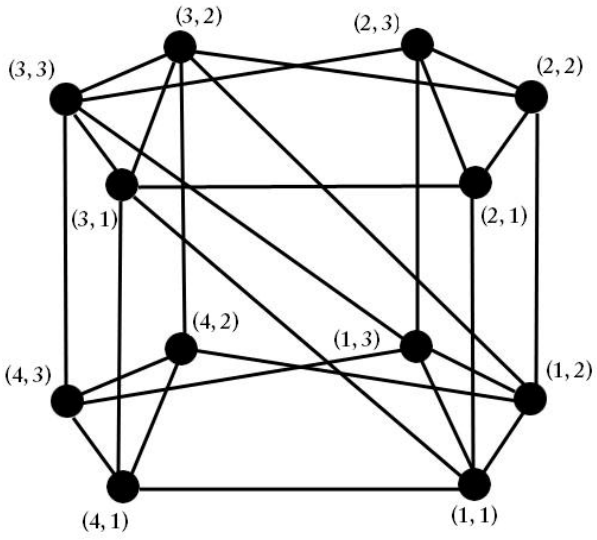

Figura 2.17 Producto cartesiano $G \times K_3$.

Se inicializa el algoritmo seleccionando un conjunto independiente de tamaño 1, es decir, formado por un único vértice. En este ejemplo el vértice seleccionado es el vértice (1,1). Así pues, *S={(1,1)}*.

El conjunto de vértices que pueden unirse a *S* sin que *S* deje de ser un conjunto independientes son los mostrados en la tabla 2.12.

Tabla 2.12 Vértices que se pueden añadir a *S* (primera iteración).

Posibles vértices (u,v) a añadir a *S*	Posibles vértices (u,v) a añadir a $S \cup \{(u, v)\}$	$\rho(S \cup \{(u, v)\})$
(2,2)	(3,3), (4,2), (4,3)	3
(2,3)	(3,2), (4,2), (4,3)	3
(3,2)	(2,3), (4,3)	2
(3,3)	(2,2), (4,2)	2
(4,2)	(2,2), (2,3), (3,3)	3
(4,3)	(2,2), (2,3), (3,2)	3

De la tabla anterior se deduce que añadiendo los vértices (2,2) o (2,3) o (4,2) o (4,3) se maximiza $\rho(S \cup \{(u, v)\})$. En este ejemplo, se selecciona el vértice (2,2) y se une al conjunto independiente *S*. Así pues $S = \{(1, 1), (2, 2)\}$.

Se repite el proceso considerando los posibles vértices que pueden incorporarse al conjunto independiente *S*. El resultado es el mostrado en la tabla 2.13.

Tabla 2.13 Vértices que se pueden añadir a *S* (segunda iteración).

Posibles vértices (u,v) a añadir a *S*	Posibles vértices (u,v) a añadir a $S \cup \{(u, v)\}$	$\rho(S \cup \{(u, v)\})$
(3,3)	(4,2)	1
(4,2)	(3,3)	1
(4,3)	Ninguno	0

Ahora se selecciona el vértice (3,3) para añadirlo al conjunto independiente S. (Aunque también sería posible elegir el vértice (4,2)). El conjunto ahora tiene tamaño 3: $S = \{(1, 1), (2, 2), (3,3)\}$

Se repite el proceso para identificar los candidatos a ser unidos al grupo independiente S y el resultado es el que se muestra en la tabla 2.14.

Tabla 2.14 Vértices que se pueden añadir a *S* (tercera iteración).

Posibles vértices (u,v) a añadir a *S*	Posibles vértices (u,v) a añadir a $S \cup \{(u, v)\}$	$\rho(S \cup \{(u, v)\})$
(4,2)	Ninguno	0

Sólo hay un candidato a unirse al conjunto S, de manera que se incluye dicho vértice y termina el proceso.

El conjunto independiente resultante, $S = \{(1, 1), (2, 2), (3, 3), (4, 2)\}$, determina unívocamente el coloreado a aplicar al grafo original G. Para determinar el coloreado hay que interpretar los dos índices de cada vértice *(u,v)* de *S* de la siguiente manera: *u* es el vértice del grafo *G* original y *v* es el color asignado a dicho vértice. Así pues, el resultado final del coloreado es el mostrado en la figura 2.18.

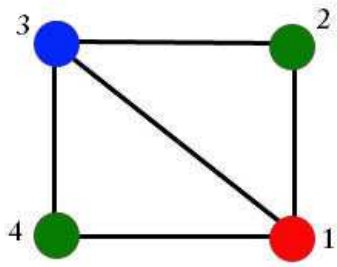

Figura 2.18 Colores asignados a cada uno de los vértices del grafo que se muestra a la izquierda de la figura. 2.16 después de aplicarle el algoritmo de *Vertex Coloring*.

2.4. Recoloreado de grafos

Los algoritmos de coloreado de grafos explicados en el apartado anterior asignan un único color a cada vértice del grafo. Sin embargo, puesto que cada color equivale a una longitud de onda, es deseable (para utilizar al máximo los recursos de la red) conocer cuál es el máximo número de colores asignables a cada vértice del grafo sin incrementar el número total de colores utilizados. Esto es lo que se define en éste trabajo como recoloreado.

Por ejemplo, suponer que se desea recolorear el grafo de la figura 2.19 (para el cual se han utilizado 3 colores en el primer coloreado). Tal y como se observa en la figura, el vértice 1 podría tener asignado el color azul (puesto que no es adyacente a ningún vértice con dicho color). El resto de vértices no pueden recibir un segundo color porque son adyacentes a todos los colores utilizados en el coloreado inicial.

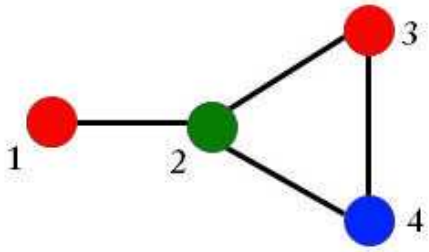

Figura 2.19 Ejemplo de grafo a recolorear.

Existen muchas maneras distintas de realizar el recoloreado de un grafo. En este trabajo se han estudiado, implementado y evaluado siete propuestas (que se describen en los apartados siguientes). Para cada una de ellas se explica el algoritmo y se muestra un ejemplo utilizando el grafo de la figura 2.20. Notar que los colores se denotan en los ejemplos mediante enteros (así, el rojo equivale al número 1, el verde equivale al número 2, el azul al 3 y el amarillo al 4).

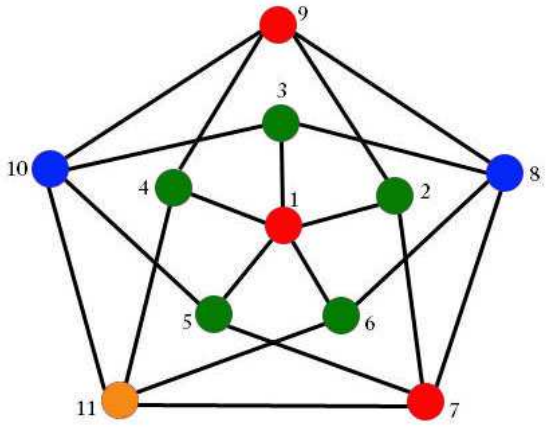

Figura 2.20 Grafo Grötzsch al que se le aplicará un recoloreado.

Las técnicas de recoloreado se han clasificado en dos grupos: ordenación estática y dinámica. En el primero, los vértices del grafo se ordenan (según distintos criterios) antes de realizar el recoloreado y dicho orden se respeta hasta finalizar el recoloreado. En el grupo de ordenación dinámica, en cambio, el orden de los vértices se modifica durante el recoloreado.

2.4.1. Recoloreado con ordenación estática

Se proponen tres técnicas de recoloreado con ordenación estática. Las tres siguen la misma técnica de recoloreado y sólo se distinguen en el criterio con el que se ordenan inicialmente los vértices del grafo.

El pseudocódigo común para los tres mecanismos de recoloreado de ordenación estática es el siguiente.

```
Requiere: Matriz de adyacencias y un grafo coloreado.

Resultado: El recoloreado del grafo.

Ordenar los vértices según el criterio elegido.

for cada uno de los vértices do
    Comprobar el color asignado en el coloreado.
    for cada uno de los vértices do
        Comprobar los colores de los vértices adyacentes.
        if queda algún color posible para asignar do
            Asignar el color al vértice.
        end if
    end for
end for
```

A continuación se describen los tres criterios considerados de ordenación de los vértices antes de aplicar el recoloreado.

2.4.1.1. Recoloreado según el identificador del vértice

En este caso, el criterio de ordenación de los vértices se establece en función del identificador de los vértices (siguiendo un orden creciente). Una vez ordenados, se intenta asignar el mayor número de colores posibles a cada vértice.

Por ejemplo, suponer que se desea recolorear el grafo de la figura 2.20. En primer lugar, se ordenan los vértices por orden creciente de identificador y se comprueba cuáles son los colores disponibles para el vértice 1 siguiendo las preguntas mostradas a continuación:

Vértice 1:

1. ¿Cuántos colores se han necesitado para colorear este grafo? 4.
2. ¿Qué color inicial tiene este vértice? Color 1.
3. ¿Qué color/colores tienen los vértices adyacentes? Los 5 vértices adyacentes tienen color 2.
4. En función de las preguntas 2 y 3, ¿qué colores se podrían asignar al vértice 1? El color 3 y el 4.

Estas preguntas son repetidas para cada vértice. Si un vértice es adyacente a otro ya recoloreado, hay que tener en cuenta todos los colores de los que dispone el vértice adyacente.

El siguiente vértice analizado por el algoritmo es el vértice 2 (con color 2). Al vértice 2 no se le puede asignar el color 1 puesto que es adyacente a los vértices 1, 7 y 9 (todos con el color 1 asignado en el coloreado inicial). Tampoco se le puede asignar el color 3 ni el color 4 (ya que ambos colores le han sido asignados al vértice 1 en el paso anterior del recoloreado). El proceso se repite hasta que todos los vértices han sido analizados.

La tabla 2.15 muestra el resultado final del recoloreado según la técnica basada en el identificador del vértice.

Tabla 2.15 Colores asignados a los vértices del grafo de la figura 2.20 después de aplicarle un recoloreado por identificador.

Vértice	Color inicial	Colores
1	1	1,3,4
2	2	2
3	2	2
4	2	2
5	2	2
6	2	2
7	1	1
8	3	3,4
9	1	1
10	3	3
11	4	4

2.4.1.2. Recoloreado según el grado del vértice

En este caso, antes de realizar el coloreado, se ordenan los vértices en función del número de vértices adyacentes (en orden decreciente). En caso de igualdad, prevalece el vértice con menor identificador. Una vez ordenados los vértices, se intenta asignar el mayor número de colores posibles a cada vértice.

```
Requiere: Matriz de adyacencias.

Resultado: Vector con el orden para recolorear.

for cada uno de los vértices do
    if un vértice tiene menor grado que el siguiente do
        Intercambiar dando mayor preferencia al vértice
        de mayor grado.
    end if
end for
```

```
for cada uno de los vértices do
    if un vértice tiene mismo grado que el siguiente do
       if ese vértice tiene mayor identificador que el
       siguiente do
          Intercambiar dando mayor preferencia al vértice
          de menor identificador.
       end if
    end if
end for
```

Utilizando el grafo de la figura 2.20 como ejemplo, la tabla 2.16 muestra el orden de los vértices para realizar el recoloreado y el resultado final del mismo.

Tabla 2.16 Resultado del recoloreado del grafo de la figura 2.20 utilizando el algoritmo basado en el número de conexiones.

Vértice	Color inicial	Conexiones	Colores
1	1	5	1,3,4
7	1	4	1
8	3	4	3,4
9	1	4	1
10	3	4	3
11	4	4	4
2	2	3	2
3	2	3	2
4	2	3	2
5	2	3	2
6	2	3	2

2.4.1.3. Recoloreado según el grado de saturación

En este caso, antes de realizar el recoloreado se ordenan los vértices en función del grado de saturación (en orden decreciente). En caso de igualdad, prevalece el identificador menor.

```
Requiere: Matriz de adyacencias y un grafo coloreado.

Resultado: Vector con el orden para recolorear.

for cada uno de los vértices do
    if un vértice tiene menor grado de saturación que el
    siguiente do
        Intercambiar dando mayor preferencia al vértice
        de mayor grado de saturación.
    end if
end for
```

```
for cada uno de los vértices do
    if un vértice tiene mismo grado de saturación que el
    siguiente do
       if ese vértice tiene mayor identificador que el
       siguiente do
          Intercambiar dando mayor preferencia al vértice
          de menor identificador.
       end if
    end if
end for
```

La tabla 2.17 muestra el resultado de aplicar el recoloreado según el grado de saturación sobre el grafo de la figura 2.20.

Tabla 2.17 Resultado del recoloreado del grafo de la figura 2.20 utilizando la técnica basada en el grado de saturación.

Vértice	Color	Grado de Saturación	Colores
6	2	3	2
7	1	3	1
10	3	3	3
11	4	3	4
3	2	2	2,4
4	2	2	2,3
5	2	2	2,4
8	3	2	3
9	1	2	1,4
1	1	1	1
2	2	1	2,3

2.4.2 Recoloreado con ordenación dinámica

En el grupo de técnicas de recoloreado con ordenación dinámica (en el que los vértices del grafo se reordenan en cada paso del recoloreado) se han considerado mecanismos que modifican el orden de los vértices en función del grado de saturación de los mismos.

El siguiente pseudocódigo muestra el proceso que se sigue siempre que es necesario reordenar los vértices del grafo:

```
Requiere: Matriz de adyacencias y un grafo coloreado.

Resultado: Vector con el orden para recolorear en función
del grado de saturación.
```

```
for cada uno de los vértices do
     if un vértice tiene menor/mayor grado de saturación
     que el siguiente do
          Intercambiar vértices dando mayor preferencia
          al vértice de mayor/menor grado de saturación.
     else if tiene el mismo grado de saturación que
     el siguiente do
               Intercambiarlos dando mayor preferencia al
               vértice de menor identificador.
          end if
     end if
end for
```

2.4.2.1. Esquema 1 de recoloreado según el grado de saturación

En este caso, antes de realizar el coloreado se ordenan los vértices en función del grado de saturación, (en orden creciente o decreciente, según se establezca). Una vez ordenados los vértices, se intenta asignar un determinado color a cada vértice. La reordenación de los vértices se realiza cuando se añade un color a un vértice.

```
Requiere: Matriz de adyacencias y un grafo coloreado.

Resultado: Recoloreado final según el esquema 1.

for cada uno de los colores do
    for cada uno de los vértices do
        Identificar si el color puede ser asignado.
        if se le asigna el color a algún vértice do
           Ordenar los vértices en función del grado de
           saturación.
           Volver a empezar el proceso con el mismo
           color para evitar no procesar algún vértice.
        end if
    end for
end for
```

La tabla 2.18 muestra los dos resultados, en función del tipo de orden, utilizando el esquema 1.

Tabla 2.18 Resultado del recoloreado del grafo de la figura 2.20 utilizando la técnica basada en el método 1.

Vértice	Color	Orden creciente (Resultado)	Orden decreciente (Resultado)
1	1	1,3,4	1,4
2	2	2	2,3
3	2	2	2
4	2	2	2,3
5	2	2	2
6	2	2	2
7	1	1	1
8	3	3,4	3,4
9	1	1	1
10	3	3	3
11	4	4	4

2.4.2.2. Esquema 2 de recoloreado según el grado de saturación

En este caso la reordenación se realiza únicamente después de intentar asignar cada color a todos los vértices de la lista.

```
Requiere: Matriz de adyacencias y un grafo coloreado.

Resultado: Recoloreado final según el esquema 2.

for cada uno de los colores do
    Ordenar los vértices en función del grado de
    saturación.
    for  cada uno de los vértices do
        Identificar si el color puede ser asignado.
    end for
end for
```

La tabla 2.19 muestra los dos resultados, en función del tipo de orden, utilizando el esquema 2.

Tabla 2.19 Resultado del recoloreado del grafo de la figura 2.20 utilizando la técnica basada en el esquema 2.

Vértice	Color	Orden creciente (Resultado)	Orden decreciente (Resultado)
1	1	1,3,4	1,4
2	2	2	2,3
3	2	2	2
4	2	2	2,3
5	2	2	2
6	2	2	2
7	1	1	1
8	3	3,4	3,4
9	1	1	1
10	3	3	3
11	4	4	4

CAPÍTULO 3. INTRODUCCIÓN A LA APLICACIÓN

En este capítulo se describe el funcionamiento básico de la aplicación realizada en este trabajo para el estudio y comparativa de los diferentes algoritmos de coloreado de grafos explicados en el capítulo anterior.

La aplicación ha sido realizada con el lenguaje de programación de Java y sobre el entorno de desarrollo Eclipse. Eclipse es una potente y completa plataforma de programación, desarrollo y compilación para realizar tareas tales como web, programas en C/C++ o aplicaciones Java. Para desarrollar la aplicación, se ha utilizado Eclipse junto con el compilador de Java JDK 6 (*Java Development Kit*) y con la máquina virtual de Java JRE 1.6 (*Java Runtime Environment*). La interfaz gráfica se ha realizado sobre SWT (*Standard Widget Toolkit*) para el cuál se ha tenido que añadir el package extra org.eclipse.swt.

3.1. Descripción del funcionamiento básico de la aplicación

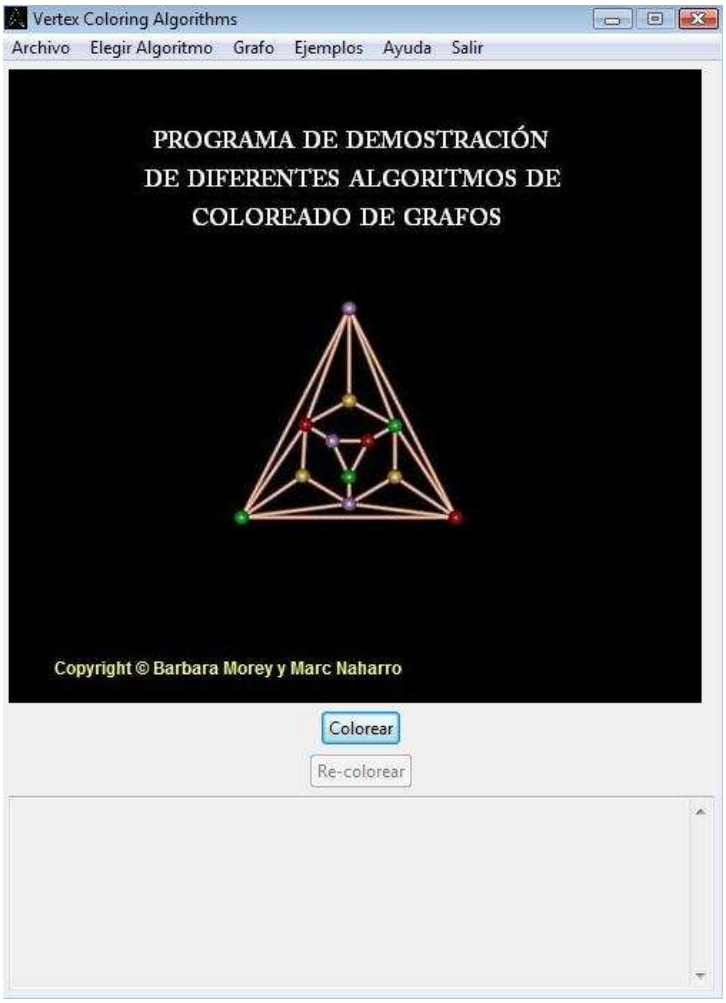

Figura 3.1 Pantalla principal de la aplicación.

El programa realiza el coloreado de un grafo previamente seleccionado o creado mediante los siguientes pasos:

1. Selección del grafo que se quiere colorear. El usuario dispone de 2 opciones a la hora de realizar este paso:

    - Elegir un grafo de entre los 20 ejemplos disponibles, desde la pestaña *Ejemplos* → *Elegir Ejemplos*.

    - Crear su propio grafo desde la pestaña *Grafo* → *Crear Grafo*.

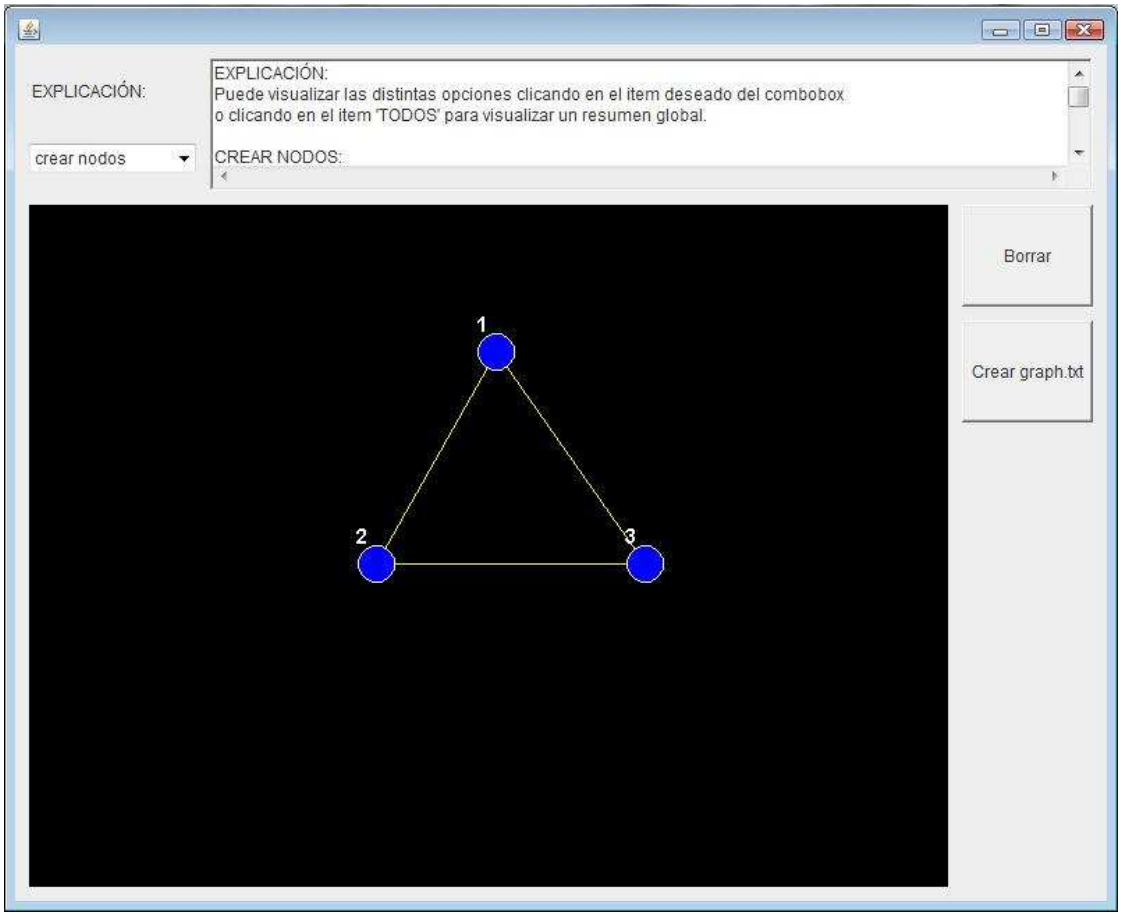

Figura 3.2 Ejemplo pantalla creación de un grafo.

Este paso genera un archivo de entrada donde se define la matriz de adyacencias del grafo a estudiar (*graph.txt*) y que será el fichero a utilizar a la hora de realizar el coloreado.

2. Selección del algoritmo con el que se va a realizar el posterior coloreado. A través de la pestaña *Elegir algoritmo* se accede a un menú desplegable con las 6 opciones disponibles.

Estos dos primeros pasos (seleccionar el grafo y seleccionar el algoritmo de coloreado) se pueden invertir, pero son necesarios ambos para poder realizar el coloreado.

3. Coloreado del grafo mediante el algoritmo elegido. El coloreado se ejecuta al pulsar el botón *Colorear* de la interfaz gráfica. Tras el coloreado se generan 3 archivos de salida: *solución.txt*, *solucionNodoColor.txt*, *solucionConexionNodoColor.txt* (cuyo contenido se especifica en el siguiente apartado).

4. Recoloreado del grafo. Una vez coloreado el grafo, se puede ejecutar el recoloreado mediante el botón *Recolorear*. Al pulsarlo, se muestran las siete opciones de recoloreado disponibles: por identificador de vértice, por grado y por grado de saturación. De este último tipo son 5 los posibles recoloreados a realizar, dependiendo del momento en el que se realiza el ordenado de los vértices. Éstos son detallados en la pestaña *Ayuda* → *Recoloreado.*

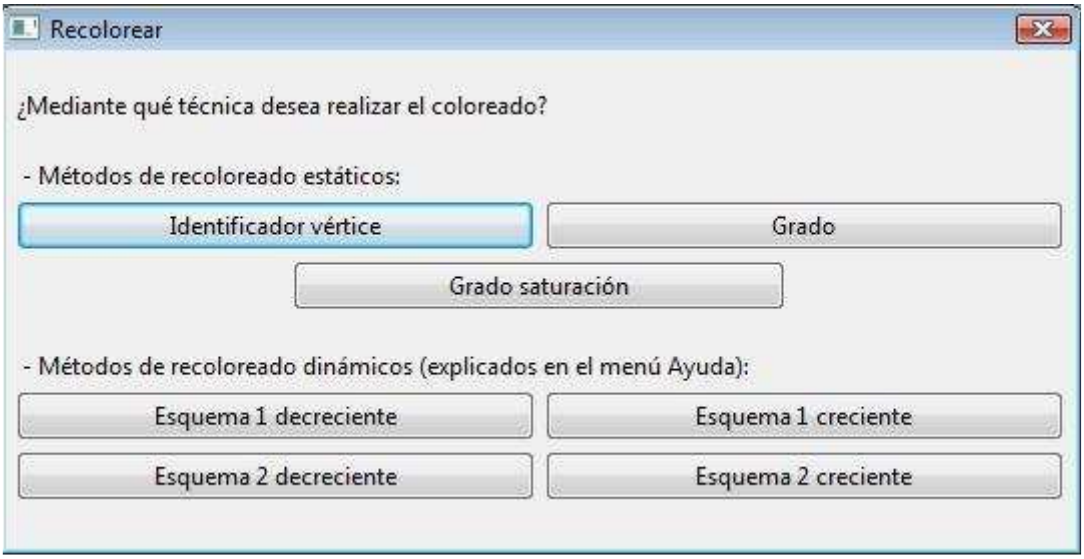

Figura 3.3 Popup del recoloreado de la aplicación.

3.2. Ficheros de entrada y salida de la aplicación

graph.txt

Es el primer archivo sobre el cual se basa el programa. Este archivo contiene la descripción del grafo sobre el que se realiza el coloreado. Su estructura sigue el formato mostrado en la figura 3.4.

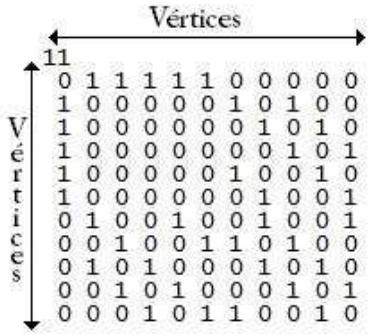

Figura 3.4 Fichero *graph.txt*.

El primer número (en la parte superior izquierda del fichero) indica el número de vértices del grafo. A continuación se detalla la matriz de adyacencias del grafo (donde un 1 en una posición de la matriz indica que dos vértices son adyacentes, mientras que un 0 indica que no lo son).

Para realizar el coloreado, la aplicación recoge este fichero y lo trata de las maneras explicadas en el capítulo anterior. Una vez realizado el proceso, a la salida se crean los 3 ficheros siguientes.

solucion.txt

Es el primero de los ficheros de salida obtenidos tras el coloreado del grafo. Su contenido es el mostrado en la figura 3.5:

```
Algoritmo de colorado (Numero de vertices): ( a1,b )( a2,b ) ... ( an,b )

Greedy First Fit Algorithm (11): ( 1,1 )( 2,2 )( 3,2 )( 4,2 ) ... ( 11,4 )
```

Figura 3.5 Fichero *solucion.txt*.

En la primera línea se muestra el nombre del algoritmo con el que se ha realizado el coloreado, seguido del número de vértices del grafo a colorear y por último una sucesión de parejas de números, donde el primero de ellos es el identificador del vértice, y el segundo es el número del color que se le ha asignado.

solucionNodoColor.txt

Es el segundo de los ficheros de salida obtenidos tras el coloreado del grafo. Su contenido es el mostrado en la figura 3.6:

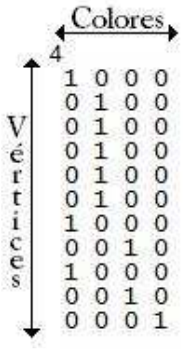

Figura 3.6 Fichero *solucionNodoColor.txt*.

Contiene la misma información que el fichero *solucion.txt* pero en otro formato. En este caso, se indica el número de colores utilizados en el coloreado (parte superior izquierda del fichero) y se representa el coloreado en formato matricial, donde los 1s son un indicador de qué color se ha asignado a cada vértice.

solucionConcexionNodoColor.txt

Es el tercero de los ficheros de salida obtenidos tras el coloreado del grafo. Su contenido es el mostrado en la figura 3.7:

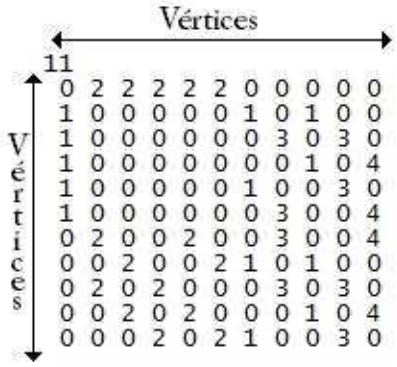

Figura 3.7 Fichero *solucionConexionNodoColor.txt*.

En este caso, el coloreado se describe utilizando la matriz de adyacencias del grafo pero sustituyendo los 1s de cada columna por el identificador del color asignado al vértice al que corresponde dicha columna. Mediante esta representación, al analizar la matriz por filas se puede determinar los colores a los que es adyacente el vértice al que corresponde dicha fila.

Recoloreado[nombre_tipo_recolorado].txt

Para realizar el recoloreado se utilizan como ficheros de entrada dos de los ficheros de salida del coloreado, los ficheros *solucionNodocolor.txt* y *solucionConexionNodoColor.txt*. Tras el recoloreado se obtiene un archivo de salida por cada recoloreado:

- *RecoloreadoIdentificador.txt*
- *RecoloreadoConexiones.txt*
- *RecoloreadoSaturacion.txt*
- *RecoloreadoEsquema1Decr.txt*
- *RecoloreadoEsquema1Cre.txt*
- *RecoloreadoEsquema2Decr.txt*
- *RecoloreadoEsquema2Cre.txt*

Todos ellos con un mismo formato aunque con un nombre diferente para poder diferenciar la metodología utilizada. El formato de cualquiera de estos siete ficheros es el mostrado en la figura 3.8:

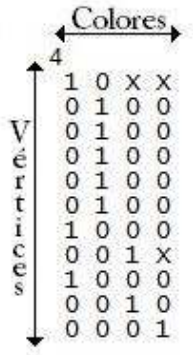

Figura 3.8 Fichero *RecoloreadoIdentificador.txt*.

En este caso, se muestran los colores asignados a cada vértice. Un 1 en la matriz identifica el color asignado por el algoritmo de coloreado inicial, mientras que las Xs identifican los colores asignados a un determinado vértice durante el proceso de recoloreado.

CAPÍTULO 4. PRUEBAS, RESULTADOS Y COMPARATIVA

En este capítulo se presentan los resultados más significativos de la comparativa entre los distintos algoritmos de coloreado y los mecanismos de recoloreado estudiados en este trabajo.

4.1. Comparativa de los diferentes algoritmos de coloreado

Para los algoritmos de coloreado, el análisis realizado se centra en los resultados obtenidos aplicando cada uno de ellos sobre el conjunto de 20 grafos incluidos como “Ejemplos” en la aplicación desarrollada en este trabajo.

4.1.1. Número de colores

El primer parámetro evaluado es el número mínimo de colores requeridos por cada uno de los algoritmos utilizados.

En la tabla 4.1 se muestran los resultados obtenidos para los grafos analizados. (En negro se muestra el número mínimo de colores utilizado por el conjunto de algoritmos comparados).

Tabla 4.1 Coloreado óptimo de los distintos algoritmos para cada grafo.

Grafo / Algoritmo	Bondy-Murty G1	Bondy-Murty G2	Bondy-Murty G3	Bondy-Murty G4	Chvátal
Vertex coloring	4	3	2	3	4
Greedy FF	4	3	3	3	4
Greedy LDO	4	3	3	3	4
Greedy MDO	4	3	3	3	4
DSatur	4	3	2	3	4
Merge	4	3	2	No solución	4
Grafo / Algoritmo	**Indian**	**Kuratowski**	**Mycielski 23**	**Mycielski 47**	**Mycielski 95**
Vertex coloring	4	2	5	6	7
Greedy FF	5	2	5	6	7
Greedy LDO	4	2	5	6	7
Greedy MDO	6	2	5	7	8
DSatur	4	2	5	6	7
Merge	4	2	5	6	7

Grafo / Algoritmo	Cube	Dodecahedron	Grötzsch	Grünbaum	Icosahedron
Vertex coloring	2	3	4	4	5
Greedy FF	2	3	4	5	4
Greedy LDO	2	3	4	5	4
Greedy MDO	2	3	4	5	4
DSatur	2	3	4	4	4
Merge	2	3	4	4	4

Grafo / Algoritmo	Octahedron	Petersen	Ramsey	Tetrahedron	Weel
Vertex coloring	3	3	7	4	4
Greedy FF	3	3	6	4	4
Greedy LDO	3	3	6	4	4
Greedy MDO	3	3	6	4	4
DSatur	3	3	6	4	4
Merge	3	3	6	4	4

Notar que el grafo *Bondy-Murty G4* no es conexo y por ello no se le puede aplicar el algoritmo de coloreado *Merge*. Sin tener en cuenta esta solución, en el resto de coloreados realizados, ha ofrecido siempre la mejor solución posible.

El algoritmo *Vertex Coloring*, no ofrece la mejor solución en dos de los casos evaluados. Este algoritmo requiere de un valor previo que condiciona su funcionamiento, el número de colores con el cual se quiere realizar el recoloredo. Para obtener estos resultados, el código tiene implementado un bucle que introduce los colores para realizar el coloreado. Este bucle se inicia en el número máximo de conexiones de un vértice e incrementa en uno este valor hasta que se encuentra una solución. Esta solución siempre es la más óptima para este coloreado ya que cuando se realizan los conjuntos independientes se ordenan de manera creciente, primero por el identificador del vértice a colorear y después por el número de color (recordar que los valores de cada vértice del grafo resultante del producto cartesiano se interpretan de la siguiente forma (a,b). Siendo ‘a’ el vértice a colorear y ‘b’ el color posible que se le puede asignar).

De las tres variantes *Greedy* analizadas, teniendo en cuenta los resultados de los 20 ejemplos utilizados, el algoritmo *Greedy LDO* es el que realiza el coloreado con un menor número de colores. Por el contrario, la variante *MDO* es la que más colores requiere. Sin embargo, es importante resaltar que la variante *FF* ofrece distintos resultado en función de la numeración de los vértices que se considere. Por ejemplo, las figuras 4.1 y 4.2 muestran el resultado de aplicar el algoritmo *FF* sobre el grafo *Bondy-Murty G3* considerando dos numeraciones distintas de los vértices. En uno de los casos (ejemplo de la derecha de las figuras) se requiere un color menos.

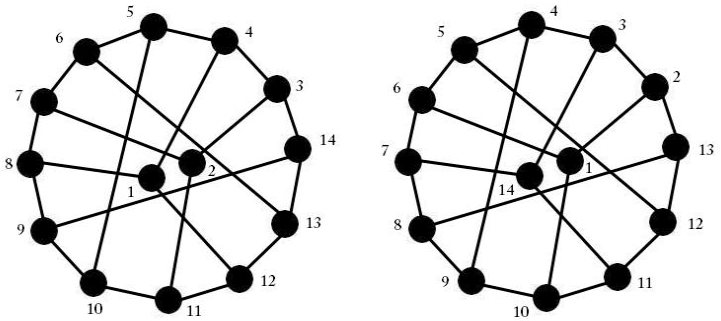

Figura 4.1 Grafo *Bondy-Murty G3*.

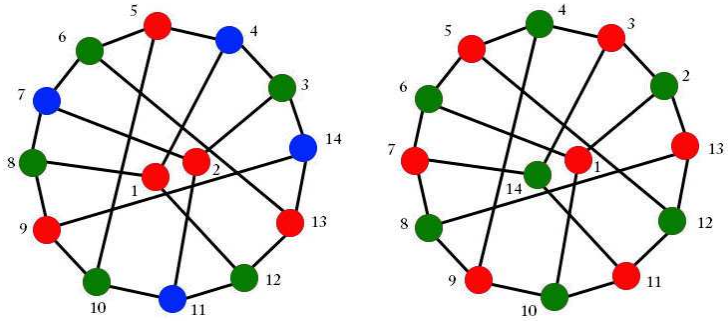

Figura 4.2 Grafo *Bondy-Murty G3 Coloreado*.

Finalmente, notar que el algoritmo *DSatur* es, de entre todos los algoritmos comparados, el que ha conseguido proporcionar la solución con el menor número de colores para todos los 20 ejemplos analizados.

4.1.2. Consumo de recursos y tiempo de ejecución

A continuación se compara el consumo de recursos y el tiempo de ejecución de los distintos algoritmos de coloreado.

La velocidad del procesador es la velocidad con la que se realizan las operaciones del código del algoritmo y la *RAM* ofrece la memoria para almacenar los recursos del programa. Puesto que el tiempo de ejecución de cada algoritmo depende del equipo utilizado, los resultados se presentan acompañados por la descripción del equipo con el que se han realizado los cálculos.

La tabla 4.2 muestra el tiempo de ejecución requerido por cada algoritmo utilizando un ordenador Pentium 4 a 1.5 GHz y 128 MB de *RAM a 400 MHz*). En la tabla sólo se muestran los resultados para 5 grafos ya que para los demás grafos analizados el tiempo de ejecución es muy pequeño. Para cada grafo, el número entre paréntesis indica la cantidad de vértices del grafo. De los 5 grafos evaluados 4 corresponden a “Ejemplos” incluidos en la aplicación. Se

ha considerado oportuno incluir el grafo restante (*Mycielski 190)* porque, al tratarse de un grafo con una gran cantidad de vértices facilita la comparativa en relación al consumo de recursos y el tiempo de ejecución.

Tabla 4.2 Tiempo, en segundos, de coloreado por algoritmo (resultados obtenidos utilizando un ordenador Pentium 4 a 1.5 GHz y 128 MB de *RAM a 400 MHz*).

Grafo / Algoritmo	Mycielski 23 (23)	Mycielski 47 (47)	Mycielski 95 (95)	Indian (30)	Mycielski 190 (190)
Vertex coloring	<1	20,5	No memory	<1	No posible
Greedy FF	<0,5	<0,5	<0,5	<0,5	<0,5
Greedy LDO	<0,5	<0,5	<0,5	<0,5	<1
Greedy MDO	<0,5	<0,5	<0,5	<0,5	<1
DSatur	<0,5	<0,5	2	<0,5	8,5
Merge	<0,5	<0,5	<1	<0,5	5,5

De los resultados de la tabla 4.2 se desprende que los tres algoritmos *Greedy* son los que menos recursos consumen y los que proporcionan un coloreado con un menor tiempo de ejecución.

Por el contrario, el algoritmo *Vertex coloring* es el que mayores tiempos de ejecución muestra. Hay que recordar que el algoritmo *Vertex coloring* requiere que se realice el producto cartesiano entre el grafo a colorear y un grafo completo con tantos vértices como colores. El resultado de dicho producto cartesiano es una matriz con un número de vértices igual al producto entre el número de vértices del grafo a colorear y el número de colores a utilizar para el coloreado.

Puesto que la aplicación no conoce el número de colores que finalmente requerirá el coloreado, comienza a colorear con un número de colores igual al número mayor de conexiones que tiene un vértice. En caso que el número de conexiones exceda de 30, se intenta colorear con 30 colores y se incrementa el número hasta que se llega a una solución. Por ejemplo, para el grafo *Mycielski 95*, el programa debe trabajar con una matriz de 2850x2850 (realizando el coloreado inicialmente con 30 colores y ofreciendo el resultado con el menor número de colores posibles, que en este caso son 7). La máquina con la que se han calculado los resultados de la tabla 4.2 no puede realizar dichos cálculos (por eso el resultado que se muestra para este grafo es *"No memory"*). A continuación se muestran los resultados utilizando una máquina más potente.

La tabla 4.3 muestra el tiempo de ejecución requerido por cada algoritmo utilizando un ordenador Pentium *Core 2 Duo* a 2 x 2.0 GHz y 2 GHz de *RAM a 667 MHz* (más potente que el anterior). En rojo se muestran los tiempos que han variado en relación a la tabla 4.2.

Tabla 4.3 Tiempo, en segundos, de coloreado por algoritmo (resultados obtenidos utilizando un ordenador *Core 2 Duo* a 2 x 2.0 GHz y 2 GHz de *RAM a 667 MHz*).

Grafo / Algoritmo	Mycielski 23 (23)	Mycielski 47 (47)	Mycielski 95 (95)	Indian (30)	Mycielski 190 (190)
Vertex coloring	<0,5	4,5	101,5	<0,5	No posible
Greedy FF	<0,5	<0,5	<0,5	<0,5	<0,5
Greedy LDO	<0,5	<0,5	<0,5	<0,5	<0,5
Greedy MDO	<0,5	<0,5	<0,5	<0,5	<0,5
DSatur	<0,5	<0,5	1	<0,5	6
Merge	<0,5	<0,5	<0,5	<0,5	2,5

La figura 4.3 muestra los datos de la tabla 4.3. Con esta gráfica se aprecia la diferencia de tiempos entre el algoritmo *Vertex Coloring* (línea en azul oscuro) y el resto de algoritmos a medida que se incrementan los vértices y las conexiones del grafo a colorear.

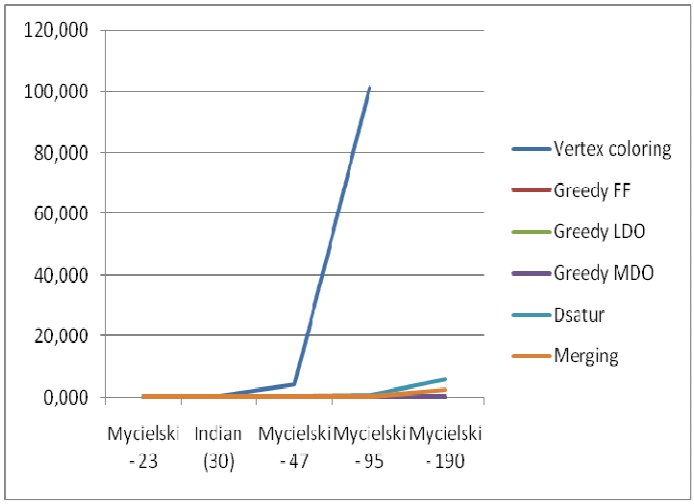

Figura 4.3 Gráfica de tiempos de resolución de grafos
(eje y:segundos, con tres decimales, eje x: grafos coloreados).

El mejor procesador ha posibilitado obtener menores tiempos de ejecución y el mayor tamaño de memoria *RAM* ha permitido solucionar el grafo de *Mycielski 95*. Aunque los 101,05 segundos requeridos para llegar a una solución distan bastante del tiempo utilizado en los demás algoritmos, que pueden considerarse menospreciables. Incluso ni con un mejor procesador ni un mayor tamaño de memoria *RAM* se ha podido solucionar el grafo *Mycielski 190* con el algoritmo *Vertex Coloring*. Esto es debido a que es el mismo Java el que se desborda al trabajar con una matriz de tales dimensiones (190*num_colores).

Los algoritmos *DSatur* y *Merge* son analizados conjuntamente ya que ambos proporcionan los mejores resultados de coloreado sobre la mayoría de grafos. Ambos algoritmos resuelven los ejemplos en buenos tiempos, ofreciendo

mejores tiempos el algoritmo *Merge* (sin considerar el grafo no conexo). Esto se debe a que el algoritmo *Merge* además de agrupar vértices en cada iteración también agrupa conexiones, dejando tras cada iteración un grafo de menor tamaño para analizar (disminuyendo los cálculos a realizar en cada iteración). El algoritmo *DSatur* en cambio, en cada iteración calcula el grado de saturación de cada vértice y los ordena en función de los valores obtenidos (manteniendo los mismos cálculos a realizar tras cada iteración).

4.1.3. Conclusiones de la comparativa realizada

Teniendo en cuenta los ejemplos analizados se puede concluir que los algoritmos *Greedy* son los más rápidos (a la hora de proporcionar una solución) aunque no siempre proporcionan un coloreado con el menor número de colores posible (siendo la variante *LDO* la que ofrece mejores resultados de entre las 3 versiones analizadas).

El algoritmo *Vertex Coloring* realiza el coloreado con un número de colores igual o inferior al de los algoritmos *Greedy* anteriores (aunque no siempre consigue minimizar dicho número). Sin embargo, el consumo de recursos que requiere su ejecución es considerablemente superior al requerido por los demás algoritmos analizados.

Finalmente, el algoritmo *Merge* es, de entre todos los algoritmos de coloreado comparados, el que mejores prestaciones ofrece (en cuanto a número mínimo de colores, tiempos de ejecución y consumo de memoria). Para grafos no conexos, el algoritmo *DSatur* es el que reúne las mejores características

4.2. Comparativa de técnicas de recoloreado

4.2.1. Técnicas de recoloreado con ordenación estática

En este apartado se compara el resultado de aplicar las tres técnicas de recoloreado con ordenación estática explicadas en el apartado 2.4.1 sobre el grafo *Grötzsch* partiendo de los cuatro coloreados mostrados en la figura 4.4.
.

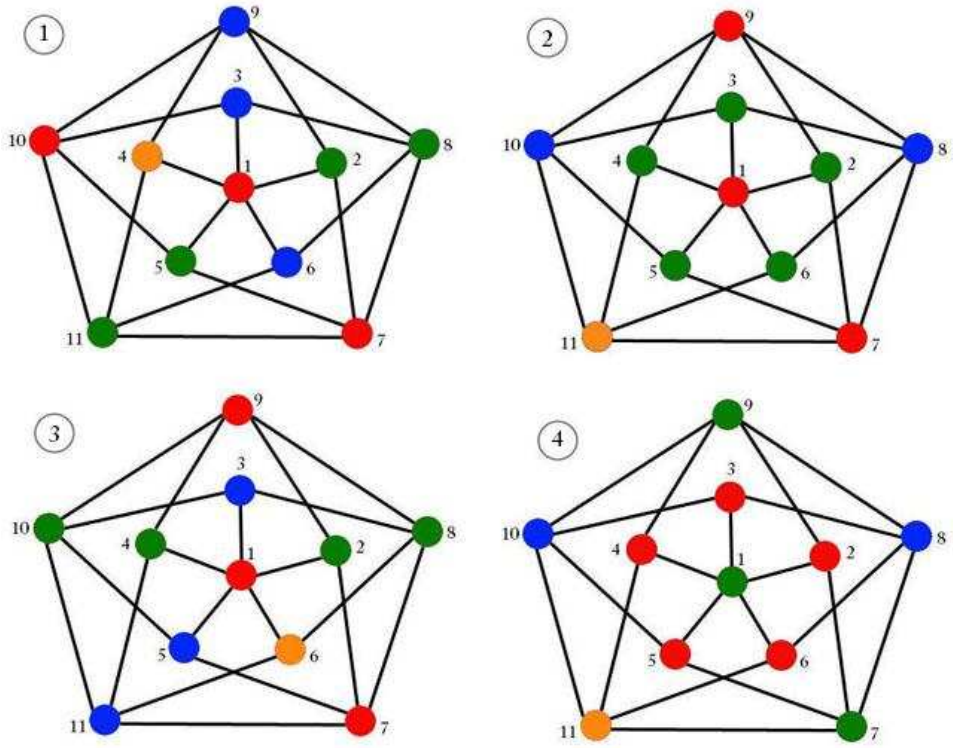

Figura 4.4 4 coloreados de *Grötzsch*.

El primer coloreado de la figura 4.4 corresponde a la solución resultante de aplicar el algoritmo de coloreado *DSatur* al grafo *Grötzsch*. El segundo coloreado es el resultado de los algoritmos *Vertex Coloring, Greedy FF* y *Merge.* El tercero es la solución del algoritmo *Greedy LDO*. El cuarto es la solución del algoritmo Greedy *MDO.*

La tabla 4.4 especifica las equivalencias entre color y número

Tabla 4.4 Equivalencias del coloreado de *Grötzsch*.

rojo	1
verde	2
azul	3
amarillo	4

Notar que los coloreados 2 y 4 de la figura 4.4 son equivalentes, simplemente están intercambiados los colores rojo (color 1) y verde (color 2), de modo que las conclusiones extraídas del análisis de las técnicas de recoloreado sobre el coloreado 2 son también aplicables al 4.

La figura 4.5 muestra el resultado de aplicar las tres técnicas de recoloreado con ordenación estática sobre el coloreado 1 de la figura 4.4. Las tres matrices, de izquierda a derecha, corresponden al resultado del coloreado por

identificador, por grado y por grado de saturación. Recordar que las filas de la matriz son vértices y las columnas son colores; el 1 de cada fila indica el color asignado por el coloreado y las Xs indican los colores asignados por la técnica de recoloreado utilizada.

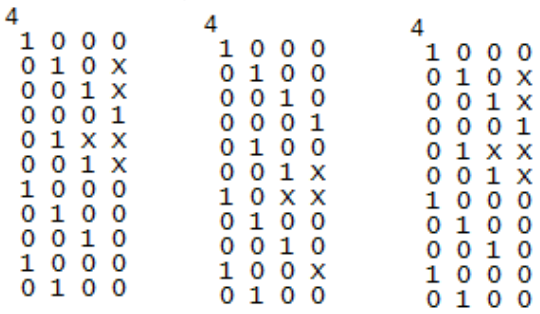

Figura 4.5 Recoloreado *Grötzsch*-1.

Las figuras 4.6 y 4.7 muestran el resultado de aplicar las técnicas de recoloreado sobre los coloreados 2 y 3, respectivamente, de la figura 4.4.

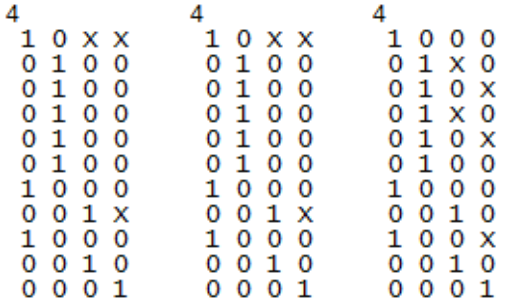

Figura 4.6 Recoloreado *Grötzsch*-2.

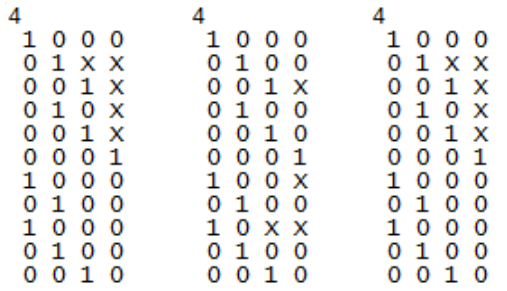

Figura 4.7 Recoloreado *Grötzsch*-3.

Ya que no se ha analizado la cantidad de tráfico de las comunicaciones de la red, se considera el mismo tráfico en todas ellas. Teniendo en cuenta este factor, será mejor asignar menos colores a una mayor cantidad de vértices que a la inversa.

Notar que de las 3 figuras anteriores (figuras 4.5, 4.6 y 4.7) el mejor y peor resultado se obtiene realizando los recoloreados sobre el coloreado 2 de la figura 4.4 (figura 4.6). Siendo el mejor resultado el obtenido aplicando el recoloreado por grado de saturación, 5 colores a 5 vértices. Por el contrario, los otros dos recoloreados (por identificador y por grado) de la misma figura asignan 3 colores a 2 vértices dando la peor asignación de colores adicionales.

Los recoloreados de las figuras 4.5 y 4.7 son muy similares, dos opciones asignando 5 colores a 4 vértices y la opción de recoloreado por grado asignando 4 colores a 3 vértices.

4.2.2. Técnicas de recoloreado con ordenación dinámica

En este apartado se compara el resultado de aplicar las 4 técnicas de recoloreado con ordenación dinámica explicadas en el apartado 2.4.2 sobre el grafo *Grötzsch* partiendo del coloreado 2 mostrado en la figura 4.4.

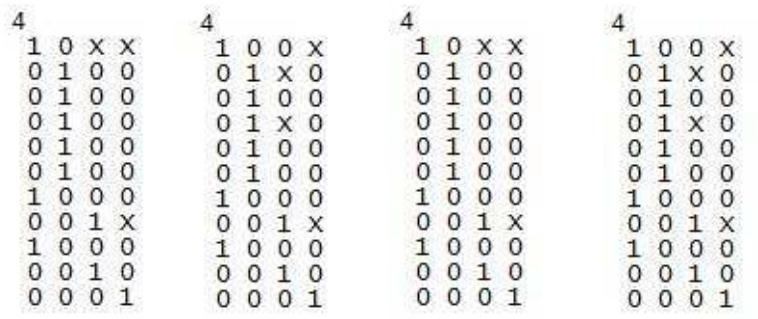

Figura 4.8 Recoloreado dinámico *Grötzsch*-2.

En la figura 4.8 de izquierda a derecha, se pueden observar los recoloreados según el esquema 1 (versiones creciente y decreciente) en el cuál los vértices del grafo se reordenan al añadir un color a un vértice y el esquema 2 (versiones creciente y decreciente) en el cuál la reordenación únicamente se realiza tras haber intentado asignar un mismo color a todos los vértices.

Notar que las matrices son iguales dos a dos, la 1 con la 3 y la 2 con la 4. Los dos esquemas de ordenado creciente han obtenido el mismo resultado y lo mismo sucede con el ordenado decreciente.

Tras analizar minuciosamente los diferentes ordenados que se llevan a cabo durante la realización de estos dos esquemas de recoloreado dinámico, se aprecia lo siguiente.

Tanto si se realiza el ordenado según el esquema 1 o el esquema 2, el resultado es el mismo. Esto se debe a que, cuando se reordena por grado de saturación al asignar un color a un vértice, los vértices que adquieren un mayor grado de saturación (mayor prioridad en el reordenado posterior) son

precisamente los que no pueden recibir el color que se está asignando (ya están conectados a ese color). Una vez llegada a esta situación, estos vértices esperan a que se les intente asignar el siguiente color, como sucede en el esquema 2.

Comparando los 2 tipos de ordenados dinámicos según si se ha realizado de manera creciente o decreciente, se muestra una mejor asignación en el segundo caso. Ni siquiera el recoloreado decreciente muestra un mejor recoloreado que el de saturación estático, que sigue siendo el mejor recoloreado.

Tras todas las pruebas realizadas se ha llegado a la conclusión que no hay un patrón fijo que marque un mejor recoloreado, sino que existe un conjunto de circunstancias como el coloreado previo y el grafo escogido que influyen en la obtención de un mejor resultado. De todas formas, la aplicación implementada en este trabajo ofrece la posibilidad de realizar cada uno de los recoloreados pulsando un botón, dejando archivos de salida de fácil comparación visual.

CAPÍTULO 5. CONCLUSIONES Y LÍNEAS FUTURAS

En este trabajo se ha estudiado el problema de la asignación de longitudes de onda en una red *OBS* considerando una estrategia de transmisión sin pérdidas. Para ello se han estudiado, implementado y comparado cuatro algoritmos de coloreado. Las tres variantes de algoritmos tipo *Greedy* que se han implementado son simples y tienen un tiempo de ejecución pequeño, sin embargo no siempre proporcionan la solución óptima. El algoritmo *VertexColoring* permite obtener mejores resultados que los anteriores, a cambio de ser el más complejo de los evaluados. El algoritmo *DSatur* es, de entre los analizados, el que ofrece mejores prestaciones (minimiza en muchos casos el número de colores utilizados en el coloreado en un tiempo de ejecución pequeño). Sin embargo, para grafos conexos, es superado por el algoritmo *Merge*.

La aplicación desarrollada para realizar el coloreado de grafos incluye 20 grafos pre-programados para facilitar la evaluación de los distintos algoritmos. También permite introducir gráficamente cualquier grafo y colorearlo. Además, el programa ofrece la opción de recolorear el grafo resultante para asignar el máximo número de colores a cada vértice (partiendo de un coloreado determinado). El resultado final del recoloreado depende del coloreado de partida y de la estructura del grafo, siendo difícil determinar cuál de las técnicas evaluadas ofrece mejores resultados.

Como posible línea de trabajo futura, sería interesante ampliar el programa para permitir al usuario obtener el resultado de la asignación de longitudes de onda introduciendo directamente la red y las comunicaciones definidas en ella. Automatizando así el proceso de generación del digrafo de restricciones (y el grafo derivado de él sobre el que se realiza el coloreado). Además, se podría incluir la matriz de tráfico como variable de entrada, lo cual supondría restricciones extra en la asignación de longitudes de onda.

BIBLIOGRAFÍA

[1] Germán, "*Estadísticas globales de Internet*", disponible en: http://www.iccc.es/2008/02/29/estadisticas-globales-de-internet/ [última fecha de acceso: enero 2009]

[2] Battestilli, T. and Perros, H., "Optical burst switching for the next generation Internet", *IEEE Potentials*, 23(5), 40-43 (2004)

[3] Gauger, C., Dolzer, K., Spätch, J. and Bodamer, S., "Service differentiation in optical burst switching networks", *ITG-Fachtagung Photonische Netze*, (2001)

[4] García, C., "*Introducción a OBS*", disponible en: http://carlos.garciaargos.com/2006/11/23/introduccion-a-obs/ [última fecha de acceso: enero 2009]

[5] Danielsen, S.L., Hansen, P.B. and Stubkjaker, K.E., "Wavelength conversion in optical packet switching", *Lightwave Technology*, 16(12), 2095-2108 (1998)

[6] White, J.Z., Le Vu, M.H. and Zukerman, M., "A framework for optical burst switching network design", *IEEE Communications Letters*, 6(6), 268-270 (2002)

[7] Chen, Y., Qiao, C. and Yu, X., "Optical burst switching: a new area in optical networking research", *IEEE Network*, 18(3), 16-23 (2004)

[8] Venkatesh, T., Sujatha, T.L. and Murthy, C.S.R., "A Novel Assembly Algorithm for Optical Burst Switched Networks Based on Learning Automata", *Lecture Notes in Computer Science*, 4534, 368 (2007)

[9] Yoo, M. and Qiao, C., "A novel switching paradigm for buffer-less WDM networks", *Optical Fiber Communication Conference*, 3, (1999)

[10] White, J., "Optical Burst Switching for Next Generation Transport Networks", *Masters-To-Phd Conversion Report*, (2002)

[11] Rosberg, Z., Zalesky, A., Vu, H.L., Zukerman, M., "Analysis of OBS networks with lmited wavelength conversion", *IEEE/ACM Transactions on Networking*, 14(5), 118-1127 (2006)

[12] Teng, J. and Rouskas, G.N., "A Comparison of the JIT, JET, and Horizon Wavelength Reservation Schemes on A Single OBS Node", *Proc. Of the First International Workshop on Optical Burst Switching*, (2003)

[13] Yoo, M. and Qiao, C., "Just-enough-time (JET): a high speed protocol for bursty traffic in optical networks", *IEEE/LEOS Technologies for a Global Information Infrastructure*, (1997)

[14] Lambert, J., Van Houdt, B. and Blondia, C., "A preventive conversion mechanism for conflict resolution in optical burst switched networks", *Procedings of 10th Conference on Optical Network Design and Modeling*, (2006)

[15] Vokkarane, V.M. and Jue, J.P., "Segmentation-Based Nonpreemptive Channel Scheduling Algorithms for Optical Burst-Switched Networks", *Lightwave Technology*, 23(10), 3125-3137 (2005)

[16] Rodríguez, E. and Agustí-Torra, A., "Mecanismes de resolució de contenses per a xarxes de commutació òptica de ràfagues (OBS)", *Treball fi de carrera* (2005)

[17] Klusek, B., Murphy, J. and Barry, L., "New Fiber Delay Line Usage Strategy in Optical Burst Switching Node"

[18] Gauger, C.M., Köhn, M., Scharf, J., "Comparison of Contention Resolution Strategies in OBS Networks Scenarios", Proceedings of the 6th International Conference on *Transparent Optical Networks,* (2004)

[19] Agustí-Torra, A., Cervello-Pastor, C. and Fiol, M.A., "A new Approach to Loss-Free Packet/Burst Transmission in All-Optical Networks", *Broadband Communications, Networks and Systems, 1-8* (2006)

[20] "*graph*", disponible en: http://www.nist.gov/dads/HTML/graph.html [última fecha de acceso: enero 2009]

[21] Saberi, N. and Coates, M., "WDM Bandwidth Allocation", *AAPN Technical Report*, (2004)

[22] Bang-Jensen, J. and Gutin, G., "Digraphs: theory, algorithms and applications", Birkhäuser, (2002)

[23] Brélaz, D., "New Methods to color the vertices of a graph", *Communications of the ACM*, 22(4), 251-256 (1979)

[24] Al-Omari, H. and Sabri, K.E., "New Graph Coloring Algorithms", *Am. J. Math & Stat*, 2(4), 439-441 (2006)

[25] Dutton, R.D. and Brigham, R.C., "A new graph coloring algorithm", *The Computer Journal*, 24(1), 85-86 (1981)

[26] Brualdi, R.A. and Massey, J.Q., "*Incidence and Strong Edge Color Graphs*", Discrete Mathematics, vol. 122, pp. 51-58, (1993)

[27] Zykov, A.A., "*On some properties of linear complexes*", Math. Sbornik, vol. 24(2), pp 163-188, (1949)

[28] Dharwadker, A., "*The Vertex Coloring Algorithm*", disponible en: http://www.geocities.com/dharwadker/vertex_coloring/ [última fecha de acceso: enero 2009]

ACRÓNIMOS

ATM: ***M****odo de* ***T****ransferencia* ***A****síncrona*

ABT: ***A****TM* ***B****lock* ***T****ransfer*

DSatur: ***D****egree* ***Satur****ation*

FDL: ***F****iber* ***D****elay* ***L****ines*

FF: ***F****irst* ***F****it*

IP: ***I****nterne****t*** ***P****rotocol*

ITU-T: ***I****nternational* ***T****elecommunication* ***U****nion* - ***T****elecommunication Standarization Sector*

JDK: ***J****ava* ***D****evelopment* ***K****it*

JET: ***J****ust-****E****nough-****T****ime*

JIT: ***J****ust-****I****n-****T****ime*

JRE: ***J****ava* ***R****untime* ***E****nvironment*

LDO: ***L****argest* ***D****egree* ***O****rdering*

MDO: ***M****inimum* ***D****egree* ***O****rdering*

OBS: ***O****ptical* ***B****urst* ***S****witching*

OCS: ***O****ptical* ***C****ircuit* ***S****witching*

OPS: ***O****ptical* ***P****acket* ***S****witching*

OXC: ***O****ptical* ***C****ross* ***C****onnect*

P2P: ***P****eer* ***to P****eer*

QoS: **Q**uality **O**f **S**ervice

RAM: ***R****andom* ***A****ccess* ***M****emory*

SWT: ***S****tandard* ***W****idget* ***T****oolkit*

WDM: ***W****avelength* ***D****ivision* ***M****ultiplexing*