UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

Búsqueda Rápida de Caminos en Grafos de Alta Cardinalidad Estáticos y Dinámicos

Autor:

Jesica Rivero Espinosa

Directores:

Dr. Francisco Javier Calle Gómez
Dra. María Dolores Cuadra Fernández

DEPARTAMENTO DE INFORMÁTICA

Leganés, 2011

TESIS DOCTORAL

Búsqueda Rápida de Caminos en Grafos de Alta Cardinalidad Estáticos y Dinámicos

Autor: Jesica Rivero Espinosa

Directores: Francisco Javier Calle Gómez
María Dolores Cuadra Fernández

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Vocal:

Vocal:

Secretario:

Calificación:

Leganés, de de

Agradecimientos

Parece que fue ayer cuando decidí empezar esta aventura que ha resultado ser mi tesis, o más bien podría clasificarla de novela, pues ha tenido su inicio, en el que parecía que todo era un mundo ideal en el que no podía ocurrir nada malo; su trama, en la que ha habido altos y bajos como en toda buena novela de suspense; y su desenlace, pues se ha conseguido llegar a este momento en el que tengo el trabajo acabado entre mis manos.

Pues bien, durante el trascurso de esta gran novela, han ido apareciendo gran variedad de personajes que han resultado cruciales para alcanzar el tan deseado final feliz, y que se merecen mi más sincero agradecimiento.

En primer lugar, quiero agradecer la ayuda prestada a todas aquellas personas que han tenido una influencia directa en este trabajo de tesis doctoral. Entre ellas se encuentran mis directores de tesis, Javi y Loli. Muchas gracias por haber depositado vuestra confianza en mí desde el principio y haberme metido el gusanillo de la investigación en el cuerpo. El siguiente es Pedro, que ha sido un miembro imprescindible en el desarrollo de este trabajo gracias a su visión de experto y a ser capaz de plantear nuevos puntos de vista y aplicación. Y por último está Yago, siempre dispuesto a prestarme la ayuda que necesitase así como servidores y más servidores. Muchas gracias, sin todos vosotros habría estado perdida en muchos momentos.

Otra persona importante para mí en este trabajo ha sido Paloma, a la que debo agradecer el haberme dado la oportunidad de poder empezar a formar parte del grupo LaBDA, y a su vez de SINTONIA, y el darme todas las facilidades que ha sido posible en todo momento. Además, el permitirme pertenecer a SINTONIA me ha llevado a conocer a la cantidad de buenas personas a las que da cabida este grupo: Isa, Lourdes, Isra, Jose, César, Elena, Harith, Ana... La verdad es que sois todos gente maravillosa que me habéis hecho vivir momentos increíbles.

Junto a los compañeros de SINTONIA también están los de IRIDIA, que durante mis tres meses en Bruselas me trataron como una más. Gracias a Eliseo, Manuelle, Carlo, Marco, Ali... y sobre todo gracias a Thomas por darme la oportunidad de estar esos meses allí, y enseñarme la gran cantidad de nuevas ideas que espero haber sabido reflejar en el trabajo final. Sois todos estupendos.

Y dentro del mundillo universitario está mi inigualable "GRUPO COMEDOR": Mayte, Mario, Julio, Marcos, Ana, Javi, Álvaro, Iván, Víctor, Arturo, Marce, Jesús, y Alberto (espero que no se me olvide nadie, y si es así le invito a unas cañas). Chicos, por favor, no cambiéis nunca. Jamás se me olvidarán esos momentos sacos de patatas, muñones de carne, safaris escolares,... y alguno más que no es el momento de decir. Sois de las mejores personas que

conozco, siempre me habéis dado vuestro apoyo y me habéis brindado vuestra ayuda sin pedir nada a cambio. Gracias.

Y también están mis chicas del comedor: Paqui, Loli, Conchi, Paloma,... Gracias por aguantarme y escucharme, a pesar de que siempre estáis hasta arriba de trabajo, y por tener siempre una sonrisa en la cara que hace que por un momento nos olvidemos del trabajo y desconectemos para descansar. Os habéis convertido en mis segundas mamás. Muchas gracias.

No obstante, el deseado momento en el que me encuentro ahora mismo no habría sido posible sin mi familia. Mamá, no habría podido ser quien soy sin ti. Tú me has enseñado que nunca hay que rendirse, que todo se puede conseguir si pongo todo mi esfuerzo, y es que solo tenía que mirarte para ver un ejemplo de superación a seguir. Supongo que todo el mundo se sentirá con el derecho de decirlo, pero yo estoy segura de poder decir que mi madre es la mejor madre del mundo. Ahora voy a por ti chache. Mi hermanito pequeño que me saca dos cabezas. Gracias por ayudarme en todo momento y por saber escucharme. Te quiero y ten siempre seguro que me tendrás cuando me necesites. Y por último, que no por ello menos importante, está Félix. Tu llegaste a mi vida para cubrir un hueco que había quedado vacío, y te has convertido en parte de mi familia siendo otro de los pilares en los que apoyarme ante las dificultades que me he ido encontrando, siempre dándome una visión pausada y realista. De verdad, gracias a todos por estar ahí, no cambiaría por nada la familia que tengo.

Y no se me puede olvidar dar las gracias a David. Ese chico que me robó el corazón hace ya muchos años (ya ni me acuerdo cuantos) cuando iba con mis ojos hinchados y mis ataques de histeria. Desde ese momento supe que había encontrado mi media naranja, mi punto de apoyo, la persona con la que podía compartir mis penas y alegrías. Gracias por escucharme, por ayudarme a levantarme cada día con una sonrisa y superar los momentos difíciles que he tenido a lo largo de todos estos años. Espero que sigamos compartiendo muchas cosas juntos durante muchos años (a poder ser todos los que nos queden).

Gracias a todos.

Summary

Graphs have been employed to resolve different kinds of complex problems since they first appeared. In order to do so, all the information associated to the domain should be represented by nodes and relationships among nodes (links) to subsequently apply different operations on the graph thus created. Among these, the search for paths is one of the most frequent, which allows different kinds of problems to be resolved.

Due to the frequency with which a large number of problems are resolved by applying path searches, a great number of papers have made a contribution to state-of-the-art algorithms to carry out such tasks. Nonetheless, due to the constant appearance of new requirements to be taken into account in graphs on which path searches are conducted, new proposals driven forward by the evolution of needs continue to arise, adding new nuances to the problem. Among these new characteristics, those which deserve special attention are the continuous growth of graphs, the fact that both the structure of graphs as well as the costs associated to nodes and links vary over time, and the appearance of new typologies arising from the size of graphs, as in the case of *Small-World Networks*. Also, as there are a large number of real-time or time-limited applications, this parameter is given priority above obtaining paths having the lowest possible cost, it being sufficient that their quality is within a specific margin with respect of the optimum.

After having conducted a study of the state of the art, it was found that there are no papers on searching for paths quickly on large generically structured dynamic graphs. Hence, this doctoral thesis work is setting out a proposal based on Ant Colony Optimisation (ACO) algorithms to cover this gap and its objectives can be summed up as follows: short response times, large dynamic graphs and a generic typology.

The fact of having chosen this kind of algorithms as a basis for the proposal is due to their proven capacity to adapt to dynamic environments, as well as on their efficiency in finding paths between nodes. Nevertheless, this type of algorithm has usually been applied to graphs having several thousand nodes at most, as it does not behave properly in larger graphs: the ants fail to find the destination or, should they do so, the quality of the path obtained is way below the optimum value. In order to allow for the use of larger graphs, some proposals incorporate the graph's pre-processing, so that the search is conducted on fragments of the main graph instead of on the complete graph. The problem with such pre-processing is that it entails a loss of adaptability characteristic of this kind of algorithm. Once a change has come about, the search has to be stopped and has to wait for the entire pre-processing to be executed once more.

In order to put forward an algorithm capable of meeting all the objectives analysed above, the proposal contained herein incorporates a path search aid which in no way modifies the structure of the graph and which is based on the way animals manage to reduce the search space to find a food source by using their sense of smell. In order to achieve this, what have been called *Food Nodes* and *Smell of Food*, have been included in the algorithm, so that the ACO (*Ant Colony Optimisation*) algorithm has become the *SoSACO* (*Sense of Smell ACO*) algorithm.

Along with this aid, it is necessary to select the best storage medium to process the massive amount of information to be used in the work, in addition to allowing concurrent access to such information. Both of these characteristics are fully covered through the use of a secondary support, more specifically a Database Management System, which allows powerful information management to be added, thereby improving access times. Furthermore, the use of a database management system will allow the problem of searching for paths in large graphs to be divided into two, making the task of searching for a path independent from the task of handling a massive amount of data. The management system will be in charge of the latter and allow this work to only deal with finding a search algorithm which is suitable for large dynamic graphs.

The stages around which the proposal is structured are explained below:

- **Choice of Food Nodes**: Choosing a series of *Food Nodes* among all the graph's nodes based on the frequency with which the node is used as a path's end/init node; on the basis of the graph's distribution, allowing the graph to be divided into areas; on the basis of the node's degree; or on the basis of the frequency of transit. Once these have been chosen, they can be used as meeting points for ants (should the food node not be the destination node), so that when an ant reaches one of them it can directly obtain the overall path linking both sections if its destination node is the node of origin of another ant that has previously reached the food node.
- **Smell of Food**: The function of this new parameter will be to mimic the smell given off by any food source in nature, creating an area around it within which it is possible to know where the food source is to be found through the sense of smell. This characteristic has nothing to do with pheromone. While the latter is used by ants to communicate indirectly among themselves, smell will only be used as an aid in their search.
- **Initial Dispersion of Smell**: The dispersion of the smell from *Food Nodes* to the nodes around them, so that the strongest smell will be found at food nodes and the nodes around them will be assigned a quantity which will progressively decrease as one moves further away from the source of the smell. Additionally, a threshold will be set from which the smell will no longer be assigned to nodes, thereby limiting the size of areas of smell. This means that it is necessary to strike a balance between the number of food nodes and the threshold.

- **Radial Dispersion of Smell**: The radial dispersion of the smell of food will come about when a food node is used to go from the starting node to the end node. This aims to mimic the fact that when a real ant gathers a piece of food, such food has a smell which will be dispersed along the path used by the ant to transport it. This fact means that the initial size of the areas of smell around food nodes will be rapidly extended, especially at frequently transited nodes thanks to the different searches made by ants. The smell will be dispersed, decreasing in value depending on the distance to the food node. In this case, the only stopping condition will be that the smell to be assigned has to be greater than zero.

As a result of this entire process, the ant will search for its destination node by making use of the pheromone trail left by other ants in a probabilistic fashion, whilst maintaining its capacity to adapt to changes thanks to this random process. If it comes across a trail of smell during the search, it can use it to go to the food node (following the increasing trail of smell) and, if it is not the destination, to use it as a meeting point.

From the explanation set out above, it can be seen that the addition made to the algorithm does not change either the graph's structure or the way ants usually go about doing their searches. Thus, if a change comes about which affects the smells or involves having to disperse them once again or deletes an already created area, the ants will not be interrupted and will be able to carry on with their search while whatever may be necessary to restore the smells is being done.

Additionally, given that finding the path solely by using the trail of smell involves following it in an increasing direction, the first approximation to be found will be shown rapidly, thereby meeting the priority objective set for this thesis: namely reducing response times.

In order to meet the remaining objectives and to greatly reduce the time employed to obtain the first response, the proposal gradually evolved through several stages at which values were set to determine the algorithm's parameters (thanks to the execution of a series of experiments over huge graphs), as well as by incorporating other new parameters which would enable the problems detected in preceding stages to be solved and new constraints to be included. In other words, an incremental iterative development method was followed.

The algorithm's stages of evolution where based in the application of the algorithm to huge static graphs with a single food node, to huge static graphs with several food nodes, to huge dynamic graphs with several food nodes, and to huge static graphs with several food nodes but with a pre-processing of paths between selected food nodes and with a paths' maintenance (applying the *SoSACO* algorithm running parallel to the path searches for the services

requested) in order to further improve response times and, in a secondary way, the quality of the paths.

After completing all these stages of evolution, and studying the results of the experiments in each one, it can be stated that the proposed algorithm is capable of reducing the rate of lost ants and the response time, of adapting to changes in the graph, and of obtaining paths with quality within a specific margin with respect of the optimum when it executes over huge graphs. In other words, all the objectives initially set for this doctoral thesis have been met.

Resumen

Los grafos han sido empleados en la resolución de problemas complejos de distinta índole desde su aparición. Para ello, toda la información asociada al dominio debe ser representada mediante nodos y relaciones entre nodos (enlaces), y posteriormente aplicar diversas operaciones sobre el grafo creado. Entre todas ellas, la búsqueda de caminos es una de las más frecuentes y permite resolver problemas de distinta naturaleza.

Debido a la frecuencia con la que se resuelven gran cantidad de problemas aplicando búsquedas de caminos, existen multitud de trabajos que han aportado al estado del arte algoritmos para realizar dicha tarea. Sin embargo, debido a la constante aparición de nuevos requisitos a tener en cuenta en los grafos sobre los que se realizan las búsquedas de caminos, siguen surgiendo nuevas propuestas impulsadas por la evolución de las necesidades, que añaden nuevos matices al problema. Entre estas nuevas características, merecen especial atención aquellas relacionadas con el continuo crecimiento de los grafos; el hecho de que tanto la estructura de los mismos como los costes asociados a los nodos y enlaces varían con el tiempo; y la aparición de nuevas topologías derivadas del tamaño de los grafos como es el caso de las *Small-World Networks*. Además, a todo esto hay que unir el que cada vez existen más limitaciones en lo que a tiempo de respuesta se refiere, pues hay gran cantidad de aplicaciones en tiempo real, o de tiempo limitado, convirtiéndose este parámetro en prioritario y situándose por encima de la obtención de los caminos con el coste óptimo.

Después de hacer un estudio del estado del arte, se ha encontrado que no existen trabajos que busquen caminos de una manera rápida sobre grafos de alta cardinalidad (a partir de este momento *grafos vastos* por simplicidad) dinámicos con estructura genérica. Por este motivo, en este trabajo de tesis doctoral se plantea una propuesta que se apoya en los algoritmos basados en colonias de hormigas (ACO - *Ant Colony Optimization*) para cubrir este hueco, de manera que los objetivos del mismo se pueden resumir en: Tiempo de respuesta reducido, grafos vastos dinámicos, y topología genérica.

El elegir este tipo de algoritmos como base para la propuesta, se debe a su demostrada capacidad de adaptación a entornos dinámicos así como a su eficacia a la hora de encontrar caminos entre nodos. Sin embargo, este tipo de algoritmos se ha aplicado a grafos de, como mucho, unos cuantos millares de nodos, pues en grafos mayores el comportamiento presentado no es correcto: las hormigas no encuentran el destino, o en caso de hacerlo, la calidad del camino obtenido se encuentra muy por debajo del valor óptimo. Para permitir su utilización en grafos mayores, hay propuestas que incorporan un preprocesado del grafo de manera que la búsqueda se realiza en fragmentos del grafo principal en lugar de en el grafo completo. El problema de este preprocesado es que hace que se pierda la adaptabilidad propia de este tipo de

algoritmos, pues cada vez que se produce un cambio, la búsqueda debe detenerse y esperar a que se ejecute de nuevo todo el preprocesado.

Con el fin de proponer un algoritmo capaz de alcanzar todos los objetivos analizados anteriormente, la propuesta presentada en este trabajo de tesis doctoral incorpora una ayuda a la búsqueda del camino que en ningún momento modifica la estructura del grafo y que está basada en la forma en la que los animales consiguen reducir el espacio de búsqueda para encontrar la comida empleando su sentido del olfato. Para ello, se ha incluido al algoritmo lo que se ha denominado *Nodos Comida* y *Olor a Comida*, pasando el algoritmo de ser ACO (*Ant Colony Optimization*) a ser *SoSACO* (*Sense of Smell ACO*).

Junto a esta ayuda, se ha seleccionado un sistema gestor de base de datos para almacenar y gestionar toda la información con la que es necesario tratar debido a la cantidad de la misma, a la necesidad de permitir un acceso concurrente a ella, y a la potente gestión de la información que proporciona y que permite mejorar los tiempos de acceso. Además, el emplear un sistema gestor de base de datos va a permitir dividir el problema de la búsqueda de caminos en grandes grafos en dos, independizando la tarea de buscar un camino de la de manejar una cantidad de datos masiva, encargándose el sistema gestor de la segunda tarea y permitiendo que este trabajo se tenga que ocupar únicamente de encontrar un algoritmo de búsqueda adecuado a grandes grafos dinámicos.

Con respecto a las fases sobre las que se apoya el algoritmo propuesto, se pueden resumir de la siguiente manera:

- **Elección de los Nodos Comida**: Se elegirán de entre los nodos del grafo basándose en la frecuencia con la que se utiliza el nodo como nodo extremo de un camino; en función de la distribución del grafo, permitiendo que el mismo quede divido en zonas; en función del grado del nodo; o en función de la frecuencia de tránsito del nodo. Una vez elegidos, podrán emplearse como puntos de encuentro de hormigas (en caso de que el nodo comida sea distinto al nodo destino), de manera que cuando una hormiga llegue a uno de ellos, si tiene como nodo destino el origen de otra hormiga que alcanzó dicha fuente de alimento con anterioridad, podrá obtener directamente el camino global uniendo ambos tramos.
- **Olor a Comida**: Este nuevo parámetro tendrá como función simular el olor que desprende cualquier fuente de alimento en la naturaleza creando un área alrededor de ella dentro de la cual se puede saber, mediante el empleo del olfato, donde se encuentra el alimento. Esta característica no tendrá nada que ver con la feromona, ya que, mientras que esta última es empleada por las hormigas para comunicarse entre ellas de manera indirecta, el olor solo será utilizado por las hormigas en su búsqueda a modo de ayuda.

- **Dispersión de Olor Inicial**: Se dispersará el olor de los *Nodos Comida* a los nodos en torno a ellos, de modo que el olor máximo se encontrará en los nodos comida, y a los nodos en torno a ellos se les asignará una cantidad que se verá reducida según el nodo esté más alejado de la fuente de olor. Además, se fijará un umbral a partir del cual ya no se asignará olor al nodo, limitando de esta manera el tamaño de las áreas con olor y siendo necesario encontrar un punto de equilibrio entre el número de nodos comida y el umbral.
- **Dispersión de Olor Radial**: Se realizará cuando un nodo comida sea empleado para ir del nodo inicio al destino. Esto trata de imitar el hecho de que cuando una hormiga real coge un trozo de comida, el mismo tiene un olor asociado que se dispersará por el camino empleado por la hormiga para transportarlo. Este hecho implicará que el tamaño inicial de las áreas de olor entorno a los nodos comida se verá extendido rápidamente, especialmente en nodos habitualmente transitados, gracias a las distintas búsquedas realizadas por las hormigas. El olor se dispersará disminuyendo su valor en función de la lejanía al nodo comida y la condición de parada será que el olor a asignar sea cero.

Gracias a todo este proceso, la hormiga buscará su nodo destino utilizando el rastro de feromona creado por otras hormigas de manera probabilística, manteniendo su capacidad de adaptación a cambios gracias a este proceso aleatorio, y si durante la búsqueda se encuentra con algún rastro de olor, podrá utilizarlo para ir hasta el nodo comida (siguiendo el rastro creciente de olor) y, en caso de no ser el destino, emplearlo a modo de punto de encuentro.

De lo explicado anteriormente se puede apreciar que la incorporación realizada al algoritmo no modifica la estructura del grafo ni cambia la manera de buscar habitual de las hormigas, por lo que si aparece un cambio que afecte a los olores y que implique tener que volver a realizar la dispersión, o borrar un área ya creada, las hormigas no se verán interrumpidas y podrán proseguir con su búsqueda mientras se realice aquello que sea necesario para restaurar los olores.

Además, puesto que el encontrar el camino empleando el rastro de olor únicamente implica seguirlo en sentido creciente, la primera aproximación de camino encontrado será mostrada rápidamente, cumpliendo de esta manera el objetivo prioritario de la presente tesis: minimizar el tiempo de respuesta.

Con el fin de cumplir con el resto de objetivos, y disminuir en mayor medida el tiempo empleado para la obtención de una primera respuesta, la propuesta fue evolucionando a través de una consecución de ciclos en los que se fueron fijando valores para determinados parámetros del algoritmo (gracias a la ejecución de una serie de experimentos llevados a cabo sobre grafos vastos), así como incorporando otros nuevos que permitiesen solucionar problemas detectados

en ciclos anteriores e incluir las nuevas restricciones. Es decir, se siguió un método de desarrollo iterativo incremental.

Respecto a los ciclos de evolución del algoritmo, se basaron en la aplicación del algoritmo a grafos vastos estáticos con un único nodo comida, estáticos con varios nodos comida, dinámicos con varios nodos comida, y estáticos con varios nodos comida pero con un preprocesado de caminos entre los nodos comida y un mantenimiento de los mismos (aplicando el algoritmo propuesto de manera paralela a las búsquedas de caminos solicitadas) con el fin de reducir el tiempo de respuesta, y mejorar de manera secundaria la calidad de los caminos solicitados.

Después de realizar todos estos ciclos de evolución, y observar los resultados obtenidos en los experimentos llevados a cabo en cada uno de ellos, se pudo concluir que se ha conseguido un algoritmo capaz de reducir la tasa de pérdida de las hormigas así como el tiempo de respuesta del algoritmo, de adaptarse a los cambios que se puedan producir en el grafo, y de obtener unos costes de caminos dentro de un rango de calidad determinado cuando ejecuta sobre grafos vastos. Es decir, se han satisfecho todos los objetivos iniciales del presente trabajo de tesis doctoral.

Índice

Índice de Figuras

Índice de Ecuaciones

Índice de Tablas

Índice de Algoritmos

Capítulo 1 Introducción

En la actualidad existen gran cantidad de sistemas que emplean grafos de alta cardinalidad (a partir de este momento *grafos vastos* por simplicidad) dinámicos como medio de representación de su información, y que precisan resolver sobre ellos búsquedas eficaces de caminos entre cualesquiera dos nodos. Ejemplos de este tipo de sistemas son los que incluyen objetos móviles (objetos que varían su posición con el tiempo), de modo que el destino puede estar moviéndose mientras se está realizando la búsqueda [Forlizzi et al., 2000; Güting et al., 2000; Liu and Schneider, 2011]; búsquedas de trayectos en redes de carreteras teniendo en cuenta que los costes de los enlaces varían con el tiempo debido, por ejemplo, a los atascos [Montemanni et al., 2005]; o búsquedas de caminos en redes complejas dinámicas [Madduri and Bader, 2009] como pueden ser los sistemas *Peer-to-Peer* (búsqueda de un documento con una serie de características determinadas), y redes sociales (buscar caminos para llegar a una persona, o para determinar la existencia de una relación entre dos personas). Puesto que algunos de ellos tienen una alta relevancia en la actualidad, como puede ser el último por su súbito impacto en la sociedad de la información, es crucial encontrar una solución a la búsqueda de caminos en este tipo de grafos.

Por ello, en este trabajo de tesis doctoral se presentará un nuevo algoritmo que intentará dar solución a este problema, caracterizándose por tanto por su rápida adaptabilidad a cambios, eficacia a la hora de buscar caminos sobre grafos vastos, y por permitir gestionar el almacenamiento de los datos masivos con los que se trata de una manera eficiente.

1.1 Motivación

Desde hace tiempo han existido sistemas que han utilizado grafos para representar la información con la que tratan. Esta herramienta ha demostrado su versatilidad para estructurar y formalizar diversos tipos de conocimiento, así como para realizar consultas sobre el mismo.

Durante los años 2007/2008, y bajo el marco del proyecto SOPAT (Servicio de Orientación Personalizada y para el Turismo (CIT-410000-2007-12), financiado por el Ministerio de

Industria, Turismo y Comercio), la autora del presente trabajo de tesis doctoral utilizó un grafo con el fin de representar el mapa de un hotel y poder realizar guiados de personas dentro de él.

El grafo resultante era de pequeñas dimensiones (la cantidad de nodos por unidad de superficie era baja), y al no tener requisitos exigentes, cualquier algoritmo clásico de búsqueda de caminos era buena solución para resolver los servicios que se solicitasen. Puesto que se empleó un sistema gestor de bases de datos para almacenar el grafo, ya que había gran cantidad de información extra asociada a cado nodo/enlace y era imposible emplear la memoria principal, se optó por utilizar uno de los que proporcionaba el sistema gestor de base de datos (concretamente, el algoritmo de Dijkstra). Sin embargo, en trabajos posteriores se incorporaron nuevos requisitos que obligaban a precisar la localización espacial, por lo que el tamaño del grafo aumentó hasta alcanzar los centenares de miles de nodos. Ante estos tamaños, se comprobó que el tiempo de respuesta de los algoritmos clásicos era inadmisible para este tipo de sistemas. Por otro lado, en este sistema se precisaba una buena solución pero no necesariamente la óptima (el camino más corto), debiendo priorizar el tiempo de respuesta a la calidad de la solución.

Esto último incentivó la búsqueda de un algoritmo alternativo que permitiese realizar estas búsquedas en un tiempo limitado, priorizando dicho tiempo por encima de la calidad del camino que se pudiese obtener (aunque manteniendo ésta, en todo caso, dentro de unos márgenes razonables con respecto a la solución óptima).

Una primera opción para abordar el problema habría sido realizar un preprocesado del grafo, pero el tamaño del mismo y el número de caminos posibles eran tan elevados que el almacenamiento necesario no era eficiente. Además, el escenario que se quería modelar tenía que tener en cuenta que ciertos pasillos podían no estar disponibles en ciertos momentos, que los ascensores estuviesen ocupados, etc., de manera que se tenía que tener en cuenta un cierto dinamismo en el escenario, describiendo un problema que se representa mediante grafos dinámicos. Las soluciones basadas en preprocesado que pueden encontrarse en el estado del arte no están preparadas para ese dinamismo, y aplicarlas conlleva reinicializar el preprocesado (o parte del mismo, en el mejor de los casos) cada vez que ocurre un cambio en el grafo. Por todo ello, la investigación tomó otro rumbo enfocándose en algoritmos que sacrificasen la solución óptima a favor del tiempo de respuesta y además pudieran aplicarse a grafos vastos dinámicos.

Aquella investigación condujo al descubrimiento de soluciones similares, en algunos aspectos, a la que se propone en este trabajo, pero ninguna de ellas tiene en cuenta todas las características del problema descrito que, aunque de origen específico, presenta muchas analogías con otros problemas del estado del arte, desplegando unas posibilidades de aplicación interesantes en diversos sistemas como, por ejemplo, aquellos que representan su información

mediante grafos que incluyen objetos móviles, los que buscan caminos en redes de carreteras, o los que realizan búsquedas de caminos en redes sociales. De especial relevancia son estos últimos, porque su súbito impacto en la sociedad de la información los ha convertido en el centro de numerosos trabajos de investigación, y porque las peculiares características del problema lo hacen complementario a aquel descrito en primer lugar, convirtiéndolo en un buen candidato para acompañarlo en una evaluación que gane en generalidad topológica. Esto es así porque la estructura de las redes sociales es de tipo *Small-World* [Newman, 2003; Tang and Liu, 2010], apartándose de la clásica topología plana que caracteriza a la mayoría de los problemas de búsqueda de caminos como, por ejemplo, la búsqueda de rutas en redes de carreteras.

1.2 Objetivos

Puesto que se quiere resolver el problema de la búsqueda de caminos de una manera eficaz cuando se trabaja sobre grafos vastos dinámicos con topología genérica, el algoritmo resultante deberá cumplir una serie de requisitos. Además, habrá que aportar una solución relativa al almacenamiento de la información para que la cantidad de datos masivos asociados a estos grafos vastos no sea un problema de cara a encontrar el algoritmo de búsqueda. Por lo tanto, deberán alcanzarse los siguientes objetivos:

- Un tiempo de respuesta mínimo. Este objetivo es el prioritario, unido al siguiente, en esta tesis, ya que el no satisfacer este requisito implicaría la imposibilidad de aplicar el algoritmo propuesto a los dominios que se explicaron con anterioridad.
- Grafo de aplicación vasto (de alta cardinalidad). Hoy en día la mayoría de los escenarios sobre los que se trabaja están alcanzando grandes dimensiones, por lo que crear un algoritmo cuyo comportamiento fuese ineficiente (alta tasa de caminos sin solución o caminos con calidad baja) para el tamaño de los grafos que los modelan no sería aceptable.
- Gran capacidad de adaptación en grafos dinámicos. Es decir, cuando haya un cambio, el algoritmo deberá poder encontrar un camino alternativo de una manera eficaz, y cuya calidad esté lo más cerca posible del nuevo óptimo.
- El algoritmo deberá poder funcionar en grafos de cualquier tipo de topología, es decir, deberá tener un comportamiento lo más genérico posible, de manera que no se creen mecanismos cuyo único fin sea el de funcionar en un tipo de grafo. El incorporar la generalidad topológica como un requisito más de la propuesta junto a los anteriormente descritos (tiempo de respuesta mínimo, gran tamaño, y dinamismo), configura un objetivo ambicioso de inmediato impacto en el estado del arte por sus posibilidades de aplicación y por la inexistencia de una alternativa que contemple todos esos requisitos.

- Manejo eficaz de la cantidad de datos con la que se trabaja. Será necesario encontrar algún tipo de almacenamiento adecuado para gestionar de una manera eficiente los datos masivos con los que se quiere tratar en el problema a abordar. Además, este tipo de almacenamiento deberá permitir un acceso eficaz a los datos para no interferir en el tiempo de respuesta.
- Independencia entre la gestión de la información almacenada y el algoritmo de búsqueda. Con el fin de que el algoritmo de búsqueda sea independiente de cómo está guardada la información, así como de la cantidad de la misma, es importante encontrar un soporte de información capaz de gestionarla sin imponer restricciones al funcionamiento del algoritmo.

1.3 Aproximación a la Solución

Para satisfacer todos los objetivos marcados en el apartado 1.2, se dividió la propuesta en dos. Por un lado se tiene que resolver el problema de la gestión de los datos y por otro la resolución de servicios de búsqueda de caminos.

1.3.1 Almacenamiento y Manejo de la Información

El primer grupo de objetivos a satisfacer es el de la gestión de recursos, o manejo eficiente de la información almacenada, importante de cara a poder dividir el problema en dos (búsqueda de caminos y gestión de recursos para el almacenamiento).

Una de las gestiones más eficientes para abordar este problema en el caso de cantidades masivas de datos puede encontrarse en las bases de datos, que gracias a los SGBD (Sistemas Gestores de Bases de Datos) son capaces de tratar con la concurrencia para mantener toda la información en un estado consistente, así como de gestionar la posible distribución de la información de manera transparente a la aplicación que esté utilizando los datos que en la base de datos estén almacenados.

Las ventajas que ofrece esta tecnología se pueden agrupar en dos categorías: aquellas que también son aportadas por otras tecnologías y aquellas que no. Dentro del segundo grupo se encuentra la capacidad para realizar un almacenamiento masivo, la aportación de una escalabilidad absoluta, una total accesibilidad a la información almacenada, gestión para realizar una total compartición de los datos o una elusión de los cuellos de botella que implica el sistema centralizado gracias a la incorporación de una gestión que hace uso de todos los procesadores y sistemas de los que se disponga. A estas ventajas propias de los SGBD hay que incorporar las que también aportan otras tecnologías como son los mecanismos de búsqueda eficiente, o las estructuras físicas que, aunque se podrían aplicar a memoria principal, es en memoria secundaria donde tienen más recorrido.

Sin embargo, el principal inconveniente de aplicar un SGBD, así como cualquier otro tipo de soporte secundario, es el impacto que tiene sobre el tiempo de respuesta. No obstante, éste puede salvarse con las posibilidades del SGBD: con el conveniente afinamiento, se puede bloquear en caché lo que se necesita frecuentemente, de modo que se tienen tiempos de respuesta comparables al proceso en memoria principal.

Por todo esto, se optó por emplear un SGBD para cumplir con los dos últimos objetivos marcados en el apartado 1.2.

1.3.2 Algoritmo de Búsqueda

Se estudiaron distintas alternativas para dar una solución al problema de la búsqueda de caminos en grafos con los requisitos indicados en 1.2. Entre ellas destacaban los algoritmos metaheurísticos, y más concretamente los basados en Colonias de Hormigas (*ACO* [Dorigo, 1992; Dorigo and Blum, 2005; Dorigo and Stützle, 2004]) por ser los que mejor se adaptan al caso de búsqueda de caminos entre dos nodos de un grafo dinámico gracias a la forma en la que operan, consiguiendo rápidos tiempos de respuesta y una adaptación instantánea ante cualquier cambio producido en el grafo sobre el que trabajan. La desventaja es que a cambio introducen una pérdida en la calidad del resultado (que en algunos casos llega a ser nula). No obstante, este hecho queda en un segundo plano al ser el objetivo prioritario la minimización del tiempo de respuesta.

Puesto que estos algoritmos se aplican a grafos con pocos nodos, se propone una adaptación denominada *SoSACO* (*Sense of Smell ACO*) con el fin de satisfacer todos los objetivos marcados. La razón de este nombre se debe a que esta evolución va a estar basada en cómo los animales son capaces de encontrar una fuente de alimento utilizando el olor que desprende dicha comida, lo que permite que los animales dirijan sus búsquedas de manera exitosa, ya que tienen pistas de hacía donde moverse en lugar de elegir de manera aleatoria una dirección de entre todas las existentes.

Para simular este comportamiento, en el grafo van a crearse zonas de olor que tendrán como fuente de comida un nodo que podrá coincidir con el destino que se busca, o que ayudará a formar caminos porque hormigas que provengan de distintos nodos inicio (que coincidan con los destinos de otra) se encuentren en él. Por lo tanto, con esta adaptación las hormigas estarán dotadas de olfato, estando su comportamiento regido no solo por el rastro de feromona sino también por el olor que desprende una fuente de alimento. De esta manera, el tiempo en encontrar el camino se verá reducido, pues la búsqueda se limitará a encontrar un rastro de olor y seguirlo hasta el nodo comida. Además se mantendrá la adaptabilidad a cambios propia de la forma de buscar de ACO, ya que si los rastros de olor desaparecen durante la ejecución de una búsqueda de camino, las hormigas podrán continuar con su búsqueda siguiendo la feromona,

pues el olor a comida es simplemente una ayuda en la búsqueda que no modifica la estructura del grafo.

1.4 Estructura del Documento

La presentación del trabajo realizado para obtener el nuevo algoritmo *SoSACO* se va a realizar mediante la división de la memoria en cinco bloques: introducción, antecedentes o estado del arte, presentación de la propuesta, evaluación, y conclusiones y líneas futuras.

En el primero de los bloques se muestra la motivación que llevó a buscar un nuevo algoritmo capaz de resolver servicios de una manera rápida sobre grafos vastos dinámicos (apartado 1.1), fijando los objetivos a perseguir (apartado 1.2) y dando una idea general de la propuesta realizada para alcanzar todas esas metas (apartado 1.3).

Una vez enfocado lo que se quiere realizar, en el segundo bloque se repasarán los trabajos que se han considerado más relevantes en el contexto que enmarca la propuesta. Para ello se hará un recorrido por las diferentes soluciones que se han ido aportando, partiendo de los algoritmos clásicos (apartado 2.1) y pasando por los algoritmos orientados a grafos vastos, tanto estáticos (apartado 2.2.1) como dinámicos (apartado 2.2.2), para llegar a las metaheurísticas haciendo especial hincapié en los algoritmos ACO por los buenos resultados que presentan en grafos dinámicos (apartado 2.3).

En el tercer bloque se presentará la propuesta de tesis (Capítulo 3), donde se indicará la solución adoptada para resolver los problemas identificados en los trabajos relacionados mostrados en el segundo bloque, así como una explicación detallada de la metodología seguida para desarrollarla y una evolución de los distintos ciclos de vida de la propuesta, partiendo de un problema simple y aumentando la complejidad del mismo hasta satisfacer todos los requisitos impuestos (Capítulo 4).

La evaluación de la propuesta se presentará en el cuarto bloque (Capítulo 5), en el que se incluirán una serie de resultados para cada uno de los ciclos de evolución del bloque anterior. Además, esta experimentación se completará con la ejecución del algoritmo en un grafo extraído de un escenario real, en concreto de una red social (apartado 5.5).

Por último, en el quinto bloque se mostrarán las conclusiones extraídas del trabajo realizado enmarcándolo dentro del estado del arte, así como una serie de líneas futuras que se quieren llevar a cabo y las publicaciones que han surgido durante el desarrollo del presente trabajo de tesis doctoral (Capítulo 6 y Capítulo 7).

Capítulo 2 Estado del Arte

Son muchos los trabajos que tratan de dar una solución al problema de la búsqueda de caminos sobre grafos, tanto de un modo general como adaptado a unas características específicas como las que persigue este trabajo. Sin embargo, no se ha encontrado ninguno que aglutine todas ellas conjuntamente ofreciendo una solución completa a la necesidad enfocada. Por este motivo, se ha estimado conveniente presentar los trabajos relacionados diseccionando la problemática por los objetivos que se persiguen.

En primer lugar, se presentarán las soluciones generales, en su mayoría soluciones clásicas o extensiones de las mismas, propuestas a lo largo de la historia del Álgebra desde que Euler planteara y resolviera el problema de los siete puentes de Königsberg [Euler, 1736], introduciendo la teoría de grafos.

Alcanzada esta visión general, se sesgará el estudio en dos direcciones. Por un lado, los trabajos que de modo específico se han centrado en resolver el problema sobre grafos vastos (tratando de encontrar las soluciones existentes hasta la actualidad para el segundo de los objetivos mostrados en la introducción), y por otro lado las propuestas relacionadas con la optimización empleando algoritmos basados en colonias de hormigas [Dorigo, 1992; Dorigo and Blum, 2005; Dorigo and Stützle, 2004] por haber sido este tipo de algoritmo el que ha aportado más éxito a la extensión del problema a grafos dinámicos (tercero de los objetivos enumerados en la introducción).

El resto de características (la generalidad de la topología, el tipo de almacenamiento, y muy especialmente el tiempo de respuesta como objetivo principal) serán tratadas de modo horizontal a lo largo de este estudio, señalando la caracterización de cada solución allá donde lo permita la descripción proporcionada por los respectivos autores en sus publicaciones.

Para finalizar, se ofrecerán una serie de conclusiones que permitirán al lector hacerse una idea global del desarrollo del área y de la cobertura de las soluciones propuestas hasta la fecha, con especial énfasis en el vacío que este trabajo viene a poblar.

2.1 Búsqueda en Grafos: Una Revisión Histórica

Euler en el año 1736 describe el problema de los puentes de Königsberg [Euler, 1736], en su búsqueda de un recorrido por las cuatro zonas en que el río Pregolya divide a esta ciudad, atravesando una sola vez cada uno de sus siete puentes y regresando al punto de partida. En su resolución, el propio Euler demuestra matemáticamente que esto es imposible formulando el primer teorema sobre la teoría de Grafos. Un siglo más tarde, Guthrie plantea un problema completamente distinto, acerca de la coloración de regiones en mapas que evite que regiones adyacentes tengan el mismo color y restringiéndose a cuatro colores (de donde obtiene el nombre de "problema de los cuatro colores"). Este problema no es resuelto hasta que en 1977 Appel y Haken [Appel and Haken, 1977; Appel et al., 1977] demuestran que sí es posible, encontrando la solución con ayuda de un ordenador. En su formalización para computar el problema, definen los términos y conceptos fundamentales de la teoría de grafos moderna [Diestel, 2006]. A pesar de lo reciente de su definición, debe observarse que su aplicación se remonta ya a casi tres siglos.

La principal característica que ha impulsado la utilización de grafos ha sido que muchos problemas de difícil resolución pueden ser expresados mediante ellos y por lo tanto ser resueltos de una manera eficaz mediante la aplicación de algoritmos de búsqueda. De hecho, la mayoría de las operaciones realizadas sobre grafos tienen que ver con su recorrido desde un nodo determinado hasta alcanzar otro que satisfaga unas condiciones dadas denominado nodo destino. Es decir, la operación más demandada consiste en realizar una búsqueda de camino dentro del grafo.

Debido a la cantidad de aplicaciones que utilizaban grafos y a la necesidad de resolver los problemas de búsquedas, aparecieron gran cantidad de propuestas para dar solución a esta necesidad. Los primeros algoritmos que surgieron se centraban exclusivamente en buscar un nodo destino dentro de todo el grafo, es decir, no tenían en cuenta ningún tipo de información de costes de la red, o número de enlaces para alcanzarlo, etc. Es decir, eran algoritmos que realizaban búsquedas sin restricciones.

No obstante, estos algoritmos no eran suficientes para dar solución a todos los problemas planteados. Uno de los más importantes era la búsqueda de los nodos destino dentro de grafos ponderados. Esto implicaba cambiar la forma de búsqueda, porque ya no era suficiente con encontrar el nodo destino, sino que se debía encontrar satisfaciendo unos requisitos como, por ejemplo, que el camino entre el nodo inicio y el destino fuese el óptimo.

De las numerosas soluciones que surgieron en respuesta a estos problemas, no puede encontrarse ninguna que sea válida en casos de explosión combinatoria, es decir, en aquellos casos en los que el número de soluciones a explorar crece rápidamente con el tamaño del grafo.

Esto suponía una gran desventaja, ya que los grafos a tener en cuenta en la mayoría de los problemas que iban apareciendo tenían cientos de miles de nodos y enlaces, lo cual implicaba no poder dar solución a las peticiones de búsqueda realizadas en un tiempo acotado [Garey and Johson, 1979].

En respuesta a esta necesidad surgieron los métodos de búsqueda heurística [Kokash, 2005]. Estos métodos suelen proporcionar buenas soluciones pero no la mejor, dado que sacrifican la optimización de ésta en favor del tiempo necesario para obtenerla, ya que en muchos casos no es necesario encontrar el óptimo sino un camino con la mejor calidad posible. Para ello, estos algoritmos disponen de algún tipo de información sobre la proximidad de cada nodo visitado al nodo destino, lo que permite explorar en primer lugar los caminos más prometedores y requerir menos tiempo para mostrar un resultado.

2.1.1 Algoritmos de Búsqueda Exhaustivos

Con la aparición de las primeras formalizaciones basadas en grafos también surgieron los primeros algoritmos de búsqueda de caminos, pues es una de las operaciones más frecuente sobre ellos. En una primera fase estos algoritmos buscaban, siempre dentro de grafos conexos, el camino óptimo de manera exhaustiva (se busca de manera sistemática y objetiva, ya que no son dependientes del problema concreto que se desea resolver). Aunque esta forma de buscar una solución requiere un tiempo de respuesta y un espacio de almacenamiento altos, se van a pasar a explicar los principales algoritmos clásicos debido a que asientan los cimientos de los más avanzados.

Una vez dicho esto, dentro de todos ellos, los primeros que surgieron fueron aquellos que buscaban caminos sin tener impuesto ningún tipo de restricción. Para ello, la norma general que sigue cualquiera de los métodos que se explicarán a continuación es que no se puede dejar ningún nodo sin visitar, y que no se puede explorar más de una vez el mismo nodo. Los algoritmos más relevantes de este tipo son:

- El algoritmo de búsqueda en amplitud (que aparece explicado de una forma clara en [Cormen et al., 2001a]). Este método empieza en el nodo inicio y explora sistemáticamente todos los enlaces que parten de él para descubrir todos los nodos adyacentes, de manera que en cada nueva iteración selecciona uno de los nuevos nodos y repite este proceso hasta alcanzar el nodo destino. Aunque este algoritmo es eficaz (siempre obtiene la solución óptima) es muy ineficiente, con complejidades del orden $O(n^p)$ donde n es el número medio de sucesores y p el nivel en el cual se encuentra la solución, lo que restringe su aplicación a grafos de dimensiones reducidas.
- El algoritmo de búsqueda en profundidad (del cual puede verse la forma en la que realiza la búsqueda de una manera detallada en [Cormen et al., 2001b]) comienza su búsqueda

con el nodo actual en el nodo inicio y la pila en la cual irá guardando los nodos visitados vacía. En cada paso se tomará como nodo actual a un nodo adyacente que no esté en la pila y se insertará en ella. Al llegar a nodos sin nodos adyacentes que no estén en la pila, se volverá al nodo anterior (cima de la pila). Si se alcanza el nodo destino, se tendrá una solución en la pila, y se operará como si éste fuera un nodo infructuoso a fin de realizar una búsqueda exhaustiva que ofrezca la solución óptima, o todas las soluciones posibles (cuando esto sea necesario). En este caso, aunque la complejidad temporal sigue siendo la misma que en el algoritmo anterior, la espacial se ve reducida, ya que solo se guarda el camino recorrido hasta el momento. Es decir, la complejidad espacial toma el valor $O(p+r)$, donde p es la longitud del camino y r es el factor de ramificación.

- El algoritmo de búsqueda bidireccional [Pohl, 1971]. Este algoritmo surge ante la problemática de disponer conjuntamente de una descripción del problema y de una meta explícita. En este caso se combinan dos grafos de búsqueda distintos y se realiza simultáneamente la búsqueda del nodo destino (encadenamiento hacia adelante) y la del nodo inicio a partir de un nodo destino (encadenamiento hacia atrás) hasta que ambos procesos confluyen en algún nodo. En caso de querer obtener todas las soluciones posibles en lugar de únicamente la óptima, se aplica una variante a la búsqueda en la que el algoritmo no parará cuando las dos búsquedas coincidan en un nodo, sino que en ese caso se guardará el camino y se proseguirá la búsqueda hacía otros nodos. De esta manera, cuando ambas búsquedas hayan terminado de explorar todos los nodos, el conjunto de todas las soluciones será la concatenación de los diversos caminos registrados para el nodo inicio y destino. En lo referente a la complejidad, puesto que aplica los algoritmos anteriores en cada una de las búsquedas, y termina cuando ambas se juntan, tanto la complejidad temporal como la espacial se verá reducida, pasando a ser ambas de $O(n^{p/2})$, donde n es el número medio de sucesores y p el nivel en el cual se encuentra la solución.

Aunque estos algoritmos son capaces de encontrar caminos entre nodos tanto cuando trabajan sobre grafos ponderados como cuando lo hacen sobre grafos no ponderados, en el caso en el que se incorpora como principal requisito el hallar el camino óptimo se abre un abanico de posibilidades sobre el que se han realizado numerosas propuestas.

Uno de los algoritmos más frecuentemente utilizados en las búsquedas de caminos óptimos entre cualesquiera dos nodos de un grafo es el algoritmo de Dijkstra, que fue descubierto en 1956 y publicado en 1959 [Dijkstra, 1959], y del cual se pueden ver ejemplos claros de funcionamiento en [Ríos Insua, 1996]. La idea principal de este algoritmo consiste en explorar todos los caminos óptimos que parten del nodo inicio y que llevan al resto de nodos. Para ello, inicialmente todos los nodos no adyacentes al nodo inicio tendrán un coste infinito, mientras

que los que sí que lo sean tendrán el coste del enlace que los una. En cada iteración, se seleccionará como nodo actual el de menor coste asignado y se actualizará el coste de sus nodos adyacentes en caso de que el que tuviesen anteriormente fuese mayor que el obtenido si fuesen al nodo inicio a través del nodo actual. El algoritmo finalizará cuando todos los nodos del grafo hayan sido empleados como nodo actual. De esta manera, una vez finalizada la ejecución, cada nodo del grafo tendrá asociado el camino óptimo para llegar a él desde el nodo inicio.

A pesar de que este algoritmo siempre aporta la solución óptima, su complejidad es elevada. Considerando que N es el número de nodos del grafo, la complejidad es del orden de $O(N^2)$. Este hecho lo hace inaplicable a grafos vastos, aunque como se verá posteriormente, existen gran cantidad de trabajos orientados a incluirle modificaciones para hacer posible dicha aplicación.

Otra desventaja del algoritmo de Dijkstra es el no permitir trabajar con costes negativos en los enlaces. Con el fin de resolver esto, se creó el algoritmo de Bellman-Ford [Bellman, 1958; Ford, 1956], que es básicamente igual al algoritmo de Dijkstra pero con la principal diferencia de que permite trabajar con dicho tipo de costes.

Puesto que tanto Dijkstra como Bellman-Ford buscan el camino óptimo entre un nodo inicio y los nodos destino, si surgiese la necesidad de encontrar el camino óptimo entre todos los nodos de un grafo, se aplicaría el algoritmo de Floyd-Warshall [Floyd, 1962; Roy, 1959; Warshall, 1962], que lo único que hace es aplicar de manera reiterada cualquiera de los dos algoritmos descritos anteriormente.

Obsérvese que Bellman-Ford (y por ende Floyd-Warshall) tiene una estructura análoga a Dijkstra, presentando costes del mismo orden que hacen inválida su aplicación a grafos vastos.

2.1.2 Algoritmos Heurísticos

Focalizando el problema de búsqueda de caminos entre nodos a grafos vastos, los costes asociados a los algoritmos descritos en el apartado anterior son, por elevados, inaceptables. Esto es debido al fenómeno conocido como *explosión combinatoria*, que hace que el conjunto de soluciones a tener en cuenta crezca de manera muy rápida a medida que el tamaño del problema crece.

En la mayoría de las aplicaciones de estos algoritmos no es necesario obtener el camino óptimo, sino que cualquier solución suficientemente próxima es válida. En estos casos, se pueden aplicar algoritmos que sacrifiquen la optimización del coste de la solución en favor del tiempo de respuesta (o que busquen un compromiso entre ambos). Dentro de esta categoría se encuadran los algoritmos *heurísticos*, que incorporan conocimiento del problema (como

información sobre la proximidad de cada nodo a un nodo destino) para eludir la búsqueda exhaustiva. Esta información se extraerá de la denominada *función heurística*.

Dentro de los métodos de búsqueda heurística cabe destacar los siguientes:

- Búsqueda en Escalada: el método de escalada [Russell and Norvig, 2003] es aquel que, en su versión más habitual, elegirá aquellos nodos en los cuales, al aplicar la función heurística elegida, el resultado sea mayor que el del nodo actual. La principal ventaja de este algoritmo es que solo se necesita el valor de la función heurística obtenido en cada momento. Por otro lado, la dependencia de esta función lo convierte en un método de difícil aplicación por la complejidad de elegir la mejor opción de entre todas las posibles.
- Búsqueda Primero el Mejor [Russell and Norvig, 2003]: recorre el grafo eligiendo en cada momento el nodo que tenga el mejor valor para la función heurística seleccionada. Además, este método permite retomar caminos abandonados previamente cuando el camino se aleja de la meta. Puesto que este algoritmo tiene como objetivo encontrar el camino óptimo, la función heurística a elegir debe medir la distancia al nodo destino. Como ventajas, cabe destacar el hecho de que evita caer en mínimos locales como ocurría en el caso del método anterior, y que además asegura encontrar el camino óptimo aunque requiera más tiempo. Por el contrario, plantea una grave desventaja, y es que no tiene en cuenta la longitud del camino recorrido hasta el momento, solo se centra en mejorar el coste del siguiente paso. Esto implica que aunque se haya encontrado un camino, la longitud total del mismo no tiene por qué ser la mejor.
- Algoritmo A^* [Hart et al., 1968]: este algoritmo resuelve el problema del anterior, ya que en la función de evaluación seguida se tiene en cuenta tanto el coste del camino que se lleva recorrido como la heurística del nodo actual. Es decir, se eligen aquellos nodos que optimicen la función $f(n) = g(n) + h(n)$, donde $f(n)$ es un estimador del coste del camino si el mismo pasase por el nodo n, $g(n)$ es el coste de llegar desde el nodo inicio a n, y $h(n)$ es un estimador heurístico del coste del camino de ir desde n al nodo destino. La ventaja principal de este algoritmo es el hecho de que si se elige adecuadamente $h(n)$ (cumpliendo unos determinados requisitos), siempre se encuentra un camino solución, y además, el coste de éste siempre será el óptimo. No obstante, esto es a su vez su mayor desventaja, y es que el coste de encontrar una correcta $h(n)$ es sumamente elevado, por lo que normalmente es preferible optar por una función que permita obtener un camino estando éste dentro de un rango de costes determinado. Además, este algoritmo también presenta un coste espacial elevado.
- Estrategia Minimax [Russell and Norvig, 2003]. Este método es utilizado en la búsqueda con adversarios con el fin de ganar una partida en la cual los dos participantes realizan

movimientos de manera alternada en el juego. Para ello, el grafo es organizado a modo de árbol, de manera que los enlaces representan la información de los posibles movimientos a realizar por ambos jugadores, y los nodos representan el máximo y el mínimo de la función de evaluación. Puesto que esta estrategia requiere recorrer todas las posibles opciones de movimiento, tiene la desventaja de ser poco eficiente (requiere mucho tiempo para poder ser todo lo exhaustivo que es necesario).

- Minimax con Poda. Método de Poda α-β: esta variante del método Minimax recorrerá el árbol creado en profundidad hasta un valor prefijado, y se transmitirá a los niveles superiores del mismo la información obtenida con el fin de evitar la exploración de algunas ramas (*poda* α-β). La poda consiste en pasar en cada llamada recursiva a un nodo hijo el valor α (cota inferior de los valores buscados en la parte inferior del árbol que queda por explorar) y el valor β (cota superior de los valores buscados en la parte inferior del árbol que queda por explorar). Si el valor de α es mayor o igual que el de β, se realizará la poda de esa rama. Esta variante de Minimax permite tratar con problemas que de otro modo no serían computables de una manera eficiente gracias a la reducción considerable del número de nodos a procesar, pero sin embargo posee una gran dependencia con el ordenamiento de los nodos del árbol.

2.2 Búsqueda en Grafos Vastos

Aunque los algoritmos heurísticos consiguen resolver el problema de la búsqueda del camino existente entre dos nodos cualesquiera del grafo haciendo frente al problema de la explosión combinatoria, siguen compartiendo un defecto con los algoritmos exhaustivos: su aplicación a grafos vastos es inviable debido a que requieren gran cantidad de tiempo de respuesta así como de espacio.

Esto resulta un problema, ya que la mayoría de las aplicaciones que se encuentran en auge en la actualidad utilizan grafos vastos para poder representar toda la información con la que trabajan. Algunos ejemplos de esto pueden encontrarse en los grafos empleados en biología computacional, redes sociales, redes de comunicación, redes de carreteras, etc. Por ello, en el primer apartado se mostrará un conjunto de trabajos que tratan de dar solución a este problema existente con el tamaño de los grafos utilizando los algoritmos vistos hasta ahora, pero con diversas adaptaciones para hacerlos válidos en estos entornos.

Por otro lado, en el estado del arte se ha encontrado una creciente preocupación por los grafos dinámicos. Esto es debido a los numerosos problemas reales a los que puede aplicarse este formalismo, como son los problemas de tráfico de vehículos (redes de carreteras, circulación por ciudades, etc.), transporte de mercancías (cintas transportadoras en fábricas,

almacenes, aeropuertos, etc.), conexión en redes sociales, etc., de modo que con el fin de dar completitud a los trabajos en grafos vastos, en el segundo apartado se mostrarán también aquellos que trabajan sobre grafos dinámicos.

El último apartado se dedicará a estudiar una de las topologías que se salen de lo general y que están teniendo una amplia difusión en la actualidad: la topología *Small-World*, y es que, por su especificidad, requiere algoritmos de características especiales para adaptarse a ella.

2.2.1 Grafos Estáticos

A pesar de la gran cantidad de algoritmos existentes, tanto exhaustivos como heurísticos, todos ellos plantean una seria desventaja: cuando la búsqueda se realiza sobre un grafo vasto su utilización se hace inviable debido al tiempo de respuesta y al espacio que requieren. Con respecto a lo primero, el problema viene de que en determinadas aplicaciones es crítico reducir (o incluso minimizar) el tiempo de respuesta. En lo que concierne al espacio, puede estar limitado por dos razones: porque existan restricciones en el coste (económico) de los sistemas o en sus dimensiones (espacio físico) que conlleva a su vez restricciones en la configuración del hardware disponible y conduce a procurar soluciones capaces de operar eficientemente con memoria física reducida (propuestas presentadas en el apartado 2.2.1.1), o a utilizar otro tipo de memoria de carácter masivo como es el caso de la memoria secundaria, que permite un almacenamiento extenso en dimensiones reducidas (apartado 2.2.1.2); o porque la utilización de grandes almacenes repercute en el tiempo de respuesta, especialmente si se trata de almacenamiento secundario, por lo que deben buscarse soluciones para reducir ese impacto ya sea mediante la utilización de su ámbito tecnológico (memorias intermedias, sistemas gestores, etc.) o mediante la adaptación de los algoritmos para reducir tanto la necesidad de acceso a la información almacenada como el volumen de la misma (propuestas presentadas también en el apartado 2.2.1.2).

Con el fin de tratar con estas dificultades cuando se trabaja con grafos vastos existen gran cantidad de propuestas, como la realizada por [Zhao and Han, 2010], en la que crean un nuevo mecanismo de indexación del grafo para aumentar el rendimiento en las consultas de búsqueda de patrones (de subgrafos) en grafos vastos almacenados en bases de datos afrontando la naturaleza NP-completa del problema a resolver. Otro ejemplo, es el trabajo presentado en [Sankaranarayanan et al., 2009; Sankaranarayanan and Samet, 2009], en el cual indican que existe un gran interés en resolver consultas relacionadas con distancias dentro de un grafo (distancia entre dos puntos, cosas que se encuentran dentro de un radio de una distancia dada, los k-vecinos más cercanos, etc.) en tiempo real estando el grafo almacenado en una base de datos. Para resolver estas consultas, exponen que lo más fácil sería guardar todas las distancias, pero que esto es inviable, y por ello proponen una serie de preprocesados: agrupar los nodos en

subgrupos con un radio máximo determinado y que distan unos subgrupos de otros una distancia mínima; guardar una distancia entre cada grupo, de manera que pueda considerarse la misma distancia para todos los nodos del subgrupo (útil cuanto más lejos estén unos grupos de otros); y utilizar árboles para estructurar el espacio. Con esto consiguen dar respuestas rápidas utilizando una base de datos para almacenar el grafo con el fin de mejorar el tiempo requerido para el manejo de la información.

No obstante, estos trabajos no tienen como objetivo resolver la problemática asociada a la búsqueda de caminos, que es la operación a la que se quiere dar solución en este trabajo de tesis doctoral. Por ello, en los apartados siguientes se va a realizar un estudio de los trabajos centrados en dicha operación que se han considerado más relevantes para extraer conclusiones y valorar si las soluciones encontradas satisfacen todas las condiciones del problema que se quiere resolver. De manera que, con el fin de estructurar los trabajos existentes, se van a clasificar en función de si el almacenamiento del grafo se encuentra en memoria principal o en memoria secundaria (sistema de ficheros o bases de datos).

2.2.1.1 Algoritmos Basados en Memoria Principal

Los algoritmos de búsqueda de caminos en grafos vastos basados en memoria principal que se han encontrado en el estado del arte presentan como preocupación constante las limitaciones de espacio en memoria, lo que habitualmente conduce a preprocesados complejos de altas necesidades de cómputo. Habitualmente se trata de extensiones o evoluciones de otros algoritmos clásicos, y las configuraciones de hardware que presentan, así como otras restricciones para su aplicación, no siempre son realistas.

El primero que se presenta aquí es el propuesto por [Edmonds et al., 2006], que opera sobre grafos de decenas de millones de nodos y enlaces. Para realizar la búsqueda primero distribuyen la información a almacenar asignando a cada procesador una parte del grafo principal (éste se divide en una serie de subgrafos), de modo que posteriormente emplean el algoritmo de Dijkstra en su versión paralela a través de una librería llamada *Parallel Boost Graph Library*. Este proceso implicará que el tiempo de respuesta dependerá en gran medida del número de procesadores del que se disponga.

Otro artículo que se apoya en la versión paralela de los algoritmos clásicos es el presentado por [Madduri et al., 2009], en el cual se propone una adaptación del algoritmo Δ-*Stepping* [Meyer and Sanders, 2003] (una versión de Dijkstra paralela que incluye un preprocesado del grafo) con el fin de resolver la búsqueda del camino óptimo en grafos con billones de nodos, demostrando que, aunque los tiempos de respuesta obtenidos no son demasiado buenos, y que dependen del número de procesadores del que se disponga, el algoritmo es escalable.

Ya alejados de la adaptación de algoritmos clásicos para convertirlos en paralelos es importante destacar los trabajos realizados por el grupo de Chan (David Cheriton School of Computer Science en la University of Waterloo) por su relevancia en el área. Entre ellos, el primero que se debe destacar es el realizado con Zhang [Chan and Zhang, 2001], en el cual muestran una primera versión del algoritmo que proponen en trabajos posteriores y que permite utilizar grafos en los que el grado de los nodos es bajo y que tienen un tamaño que va desde los 161.595 nodos (y 197.861 enlaces) hasta los 2.507.774 nodos (y 3.169.730 enlaces), todos ellos representando mapas de carreteras reales. Se basa en crear un grafo jerárquico (que se almacenará también en memoria principal) a través de continuas divisiones del grafo en subgrafos. El proceso consiste en crear fragmentos y calcular los caminos más cortos entre los nodos frontera (nodos que pertenecen a dos o más fragmentos) de los distintos fragmentos y entre cada uno de los nodos frontera de dentro del mismo fragmento. Los nodos frontera se guardarán en un nivel superior, así como los caminos calculados (que serán los enlaces del nuevo nivel). Es decir, en el nivel inferior se tendría el grafo original, y en el nivel superior los nodos frontera y los enlaces que representan los caminos entre cada uno de esos nodos frontera. Si el nuevo nivel obtenido tuviese gran cantidad de nodos, se volvería a realizar el mismo proceso mencionado anteriormente (aunque, tal y como indican en el artículo, solo son necesarios dos niveles en todos los casos estudiados). En cuanto al algoritmo de búsqueda utilizado, una vez creado el grafo jerárquico, es el algoritmo de Dijkstra modificado para que pueda trabajar con el grafo fragmentado y mejorar de esta manera los tiempos de respuesta. El procedimiento a seguir es, en primer lugar, buscar en el grafo de nivel superior para que una vez se tenga el camino en ese nivel, ampliarlo con los nodos del nivel más bajo (los nodos que realmente forman el grafo con el que se trabaja).

En [Barrett el al., 2006] también se presenta un algoritmo que después del preprocesado, modificando la estructura del grafo sobre el que se trabaja, aplica el algoritmo de Dijkstra con algunas variantes. En él se trabaja con grafos de cientos de miles de nodos y se buscan caminos entre dos nodos cualesquiera con la restricción adicional de que un enlace pueda ser utilizado o no por el usuario. Por ejemplo, determinados vehículos no pueden transitar ciertas vías, como una bicicleta por una autopista. Además, define una serie de modificaciones que pueden ser de interés para hacer más rápida la búsqueda: búsqueda bidireccional basada en Dijkstra; añadir contenedores del camino más corto que indican por cada uno de los enlaces del grafo, que nodos son alcanzables a través de él con el camino óptimo; vectores de bits (que siguen prácticamente el mismo concepto que el contenedor de los caminos óptimos); y técnicas multinivel (descomponer el grafo en subgrafos colocados en distintos niveles).

Existen trabajos similares que extienden otros algoritmos de búsqueda, como los siguientes que se apoyan en el algoritmo A^*:

- Los realizados en [Delling et al., 2006a, 2009]. En este caso se trabaja con grafos con 15,4 millones de nodos (y 35,7 millones de enlaces). La forma de organizar toda esta información es dividiendo el grafo principal en subgrafos y buscar sobre esos fragmentos en lugar de sobre el grafo principal, de manera que al reducirse el espacio de búsqueda se tarda menos en dar una respuesta. Se obtienen respuestas a los caminos en un tiempo igual a 22 microsegundos de media, aunque emplean 48 horas para realizar el preprocesado. Además de fragmentar, también se reduce el número de enlaces mediante las dos siguientes técnicas de preprocesamiento: eliminar aquellos enlaces que son muy largos y que nunca se utilizan por existir alternativas más cortas, y guardar los caminos más cortos como si fuesen enlaces eliminando aquellos que formaban parte de ellos pero manteniendo los nodos que son utilizados como encrucijada de varios caminos.

- Otra versión del algoritmo indicado anteriormente es la presentada en [Sanders and Schultes, 2005] y que ha sido evaluada de distintas maneras, así como explicada en detalle cada una de sus fases en [Bast et al., 2007; Delling et al., 2006b; Sanders and Schultes, 2006, 2007]. En este caso se incluyen los siguientes elementos para disminuir los tiempos de búsqueda de caminos en grafos vastos: una jerarquía, de manera que en la búsqueda solo se puede ascender en ella, nunca bajar, y que guarda la distancia entre los vértices del nivel superior; una división de la búsqueda en dos, es decir, se parte del nodo inicio y del nodo destino y se trata de encontrar el cruce de la búsqueda (en la anterior versión solo partía de uno de los nodos); se aplica el algoritmo de búsqueda A^* dentro de la jerarquía, de tal manera que se busca en los nodos adyacentes y cuando se encuentra uno de nivel superior se asciende dentro de la jerarquía; y, por último, se utilizan unos puntos denominados *Landmarks* que indican nodos con importancia en la aplicación y que se utilizarán en las heurísticas.

- Como último trabajo dentro esta categoría es interesante destacar el realizado en [Goldberg et al., 2007], con una mejora del tiempo de respuesta basada en un preprocesado para la caracterización de los nodos. En este caso, los grafos sobre los que se trabaja son dirigidos y están formados por 20 millones de nodos, empleando un máximo de tres horas en el preprocesado y un tiempo de respuesta de unos cuantos milisegundos. En cuanto a la fase de preprocesado, se realizan las siguientes acciones: los nodos se clasifican dentro de *High Reach*, *Low Reach* o *Medium Reach* en función de si son normalmente alcanzados (si se utilizan en el camino) cerca del origen o no; se definen una serie de nodos con importancia en la aplicación (*Landmarks*) y se guardan todas las distancias de ellos a los nodos alcanzables con una tasa mayor que un valor fijado *R* y de estos a los *Landmarks*; se eliminan algunos enlaces sustituyéndoles por el camino óptimo, de manera que al disminuirse el espacio de búsqueda el tiempo de

respuesta es menor; y por último, emplean también lo que denominan *Improvement Localty*, que consiste en hacer una ordenación en memoria de los nodos de manera que aquellos que están incluidos en el grupo de nodos *High Reach* están cerca y por tanto caben en caché. Respecto a las variaciones sobre el algoritmo A^*, utilizan una función heurística equivalente para ambos sentidos del enlace, empleando la desigualdad triangular (en cualquier triángulo la longitud de uno de los lados no puede superar nunca a la suma de las longitudes de los otros dos); la búsqueda parte del inicio y del destino (A^* bidireccional); se aplica una condición de parada distinta a la de parar cuando se encuentra un nodo elegido por la búsqueda en el otro sentido; y, por último, indicar que en la fase de inicialización de la búsqueda se calculan dos *proxies* para el nodo inicio y dos para el destino, entendiendo por *proxy* aquel nodo dentro del grupo *High Reach* que se tiene más cerca.

En resumen, la característica general en todos estos algoritmos es conseguir un preprocesado que permita obtener información útil en la búsqueda y que se pueda almacenar en el menor espacio posible en memoria principal. Con el fin de superar ésta limitación, surgen una serie de trabajos basados en almacenamiento secundario, que son el objeto de estudio del apartado siguiente.

2.2.1.2 Algoritmos Adaptados a Memoria Secundaria

Una de las alternativas al almacenamiento en memoria principal es la utilización de algún tipo de soporte secundario. Puesto que este tipo de almacenamiento supera la mayoría de restricciones espaciales, el preprocesado puede ser más ligero y requiere un menor tiempo de cómputo.

No obstante, hay autores que combinan ambos tipos de almacenamiento para aumentar la eficiencia. Un ejemplo de esto es el trabajo presentado por [Ajwani et al., 2006, 2007]. En este caso trabajan con grafos de 130 millones de nodos y 1,4 billones de enlaces y, aunque utilizan almacenamiento en memoria secundaria, mantienen una parte almacenada en memoria principal. La cantidad de información que se encuentra en uno y otro tipo de memoria vendrá indicada por una serie de parámetros previamente definidos. Al igual que otros trabajos del apartado anterior, el algoritmo propuesto realiza un preprocesado que descompone el grafo en niveles partiendo desde un nodo inicio. Para ello emplea entre 2,3 y 4 horas (dependiendo del número de procesadores). Con esto, lo que intenta es reducir el número de lecturas/escrituras que es lo que más afecta a las búsquedas que se realizan sobre grafos almacenados en soporte secundario.

El algoritmo de búsqueda empleado implementa los algoritmos de Munagala y Ranade [Munagala and Ranade, 1999] y de Mehlhorn y Meyer [Mehlhorn and Meyer, 2002], y realiza

una comparación empírica entre ellos (Se puede ver más información de estos algoritmos en Katriel y Meyer [Katriel and Meyer, 2003]):

- El algoritmo de Munagala y Ranade muestra un algoritmo de búsqueda en anchura sobre gafos no dirigidos. Se basa en generar los nodos del nivel *K* a partir de los del nivel *K-1* de la siguiente manera: construye conjuntos que contienen todos los nodos adyacentes de los nodos del nivel *K-1*, ordena los conjuntos y quita aquellos que están duplicados, además de aquellos que pertenecen al nivel *K-1* o al *K-2*.
- En cuanto al algoritmo de Mehlhorn y Meyer, mejora al anterior reduciendo el estudio de los conjuntos creados (por lo que disminuye el tiempo de respuesta), y en el caso de Meyer [Meyer, 2001] considera la aplicación sobre grafos no dirigidos donde el grado de los nodos es limitado.

En lo que respecta a aquellos trabajos que solo utilizan el soporte secundario como medio de almacenamiento, destacan los realizados por el grupo de Chan, entre los que merecen especial interés [Chan and Lim, 2007; Chan and Zhang, 2007]. Aunque inicialmente trataron de resolver el problema del manejo de grafos vastos almacenando toda la información en memoria principal (ver Algoritmos Basados en Memoria Principal), debido a las limitaciones que esto imponía, pasaron a crear un algoritmo para tratar con el manejo de la información en memoria secundaria.

Al igual que en la versión en memoria principal, debido al gran tamaño del grafo, y a que el tiempo de respuesta es limitado, preprocesan el grafo inicial dividiéndolo en fragmentos, de manera que cada uno de ellos quepa en memoria principal, y una vez hecho esto se construye un grafo que contiene únicamente los nodos frontera y los enlaces igual a los caminos entre ellos calculados con Dijkstra.

En cuanto al procedimiento a seguir para resolver la búsqueda de caminos entre dos nodos dentro del grafo, el trabajo presentado consta de dos fases: búsqueda en el grafo esqueleto (aunque en los fragmentos en los que se encuentran los nodos inicio/destino del camino se busca el camino entre estos y sus correspondientes nodos fronteras utilizando toda la información del grafo original en dicho fragmento) y rellenado del camino encontrado con los enlaces y nodos que lo forman accediendo al grafo original. Para mejorar los tiempos de búsqueda de la primera fase, en el preprocesado se proponen dos alternativas de mejora:

- Calcular los límites superiores e inferiores de longitud de nodo inicio a destino pasando por cada fragmento para eliminar del cómputo aquellos fragmentos cuyo mínimo sea mayor que la máxima distancia entre inicio y destino.

- Con el fin de reducir el espacio requerido para almacenar las distancias anteriores y el tiempo de cómputo del plan anterior, en esta versión se plantea preprocesar únicamente las distancias entre fronteras separadas por *X* saltos.

Con todo este trabajo consiguen tiempos de respuesta bajos gracias a que existe gran cantidad de información preprocesada.

Junto a esta propuesta, merecen también atención las recopiladas en [Yu and Cheng, 2010], ya que se centra en aquellas propuestas de interés que intentan dar solución al problema de buscar caminos entre dos nodos (obteniendo el camino en sí, o su coste, o simplemente indicando si existe un camino) que se encuentran dentro de grafos vastos. Todos los trabajos presentados tienen como característica común el preprocesado realizado para poder resolver las búsquedas requeridas en poco tiempo, es decir, la creación de estructuras auxiliares para poder recorrer el grafo de una manera más eficaz. Como ejemplos puede mencionarse el trabajo de [Schenkel et al., 2004, 2005] que fragmenta el grafo; el de [Cheng et al., 2008] que realiza una partición organizada de manera jerárquica; el indicado en [Jin et al., 2008] consistente en realizar un árbol con los caminos entre nodos, de modo que si se calculase dicho árbol para todos los nodos se tuviese el grafo original; o el de [Cohen et al., 2002], que crea un índice denominado *2-hope cover* que indica los nodos alcanzables desde cada uno de los nodos, así como qué nodos alcanzan a cada nodo, y que ha sido mejorado en [Jin et al., 2009] con el índice denominado *3-hope cover*.

2.2.1.3 Conclusiones

Después de estudiar una parte significativa de los trabajos existentes en el área de la búsqueda de caminos en grafos vastos, se puede concluir que el preprocesado de los mismos es una técnica habitual para mejorar su manejabilidad, y hacer más eficiente su operación. Para ello, las técnicas a emplear son diversas: fragmentación del grafo en subgrafos, creación de jerarquías, árboles, etc. Además, el preprocesado que registra los resultados en memoria principal es más costoso, ya que las limitaciones de espacio conllevan la necesidad de sintetizar el conocimiento auxiliar.

En lo referente a los tiempos de respuesta, los valores medios son del orden de pocos milisegundos. Sin embargo, para que esto sea así se necesitan unos tiempos medios de preprocesado de varias horas, lo que descarta este tipo de métodos, en general, para su aplicación en problemas que se formalizan mediante grafos dinámicos.

Por lo tanto, cuando se aplica preprocesado existen dos claras desventajas: se requiere gran cantidad de espacio para almacenar toda la información resultante del preprocesamiento (que podría quedar resuelto con la utilización de bases de datos como sistema de almacenamiento), y

el tiempo de preprocesado (que, por ejemplo, en el caso de [Delling et al., 2006a, 2009] alcanza las 48 horas).

Esto último no supone ninguna desventaja si se trabaja con grafos estáticos. No obstante, aunque muchos de los problemas a resolver cumplen estas características, esto no ocurre en todas las posibles aplicaciones. Por ejemplo, en el caso de las redes ad hoc, las redes sociales, las redes de telecomunicaciones, etc. el grafo es dinámico, sufriendo cambios frecuentes (cada pocos segundos). Esto implica que el preprocesado tendría que actualizarse cada vez que se opera un cambio. Si los cambios son frecuentes, pueden ocurrir antes de que termine el preprocesado, siendo nulo el tiempo de validez de sus resultados y, por tanto, no pudiéndose aplicar los métodos descritos que se basan en él. Por otro lado, incluso si los cambios no son tan frecuentes o si estos afectan de modo parcial (local) al preprocesado, sí pueden alargar el tiempo de respuesta por la contingencia de tener que sumar a éste el tiempo de preprocesado (o parte de él).

Algunos trabajos de los anteriormente mencionados, como [Delling et al., 2006a, 2009], indican que ante cambios solo tendrían que modificar parte del preprocesamiento realizado, y que esto no requeriría mucho tiempo. No obstante, cuando los cambios se efectúan con una alta frecuencia temporal, y los mismos afectan a distintas zonas del grafo, todo el preprocesado debe ser rehecho (o al menos una gran mayoría) y el problema del manejo con grafos vastos se hace inviable.

El siguiente apartado versa sobre los trabajos que específicamente centran sus esfuerzos en ofrecer soluciones eficientes sobre grafos dinámicos.

2.2.2 Grafos Dinámicos

La búsqueda de caminos en grafos vastos dinámicos es una necesidad en la actualidad ya que, como se señala en [Nannicini and Liberti, 2008], casi todos los trabajos acerca de grafos vastos se refieren a grafos estáticos. Además, estos autores dan una definición para el término *dinamismo*, que será la empleada en este documento, y que indica que por *dinamismo* se entiende cualquier cambio en un grafo: cambio en costes, enlaces o nodos.

Una vez aclarado que se va a entender por dinamismo, al igual que ocurría en el caso de los grafos con características estáticas, en los grafos vastos con características dinámicas los trabajos pueden dividirse entre aquellos que utilizan la memoria principal para almacenar la información, y aquellos que optan por utilizar un soporte secundario de almacenamiento para poder disponer de una menor limitación espacial. Por este motivo, las propuestas se presentarán agrupadas de esa manera.

2.2.2.1 Almacenamiento en Memoria Principal

Entre todos los trabajos que se encuentran en esta área, cabe destacar en primer lugar aquellos que ya intentaban resolver el problema en estática y que han intentado adaptar sus propuestas a grafos dinámicos.

Un ejemplo de esto es [Chan and Yang, 2009], en el cual se muestra un algoritmo que reordena árboles del camino óptimo creados sobre grafos dinámicos, es decir, presenta un algoritmo capaz de llevar a cabo el mantenimiento del árbol creado para que siempre se encuentre en un estado consistente, por ejemplo, quitando enlaces del árbol porque ya no se incluyan en el camino óptimo e incluyendo otros porque sean necesarios en él en ese momento. El tipo de dinamismo que tienen en cuenta hace referencia a los cambios en los costes de los enlaces (tanto crecimiento como decrecimiento de los mismos), y hacen hincapié en que toman los cambios de manera global, no uno por uno, es decir, no consideran que los cambios ocurran de manera secuencial. Respecto al algoritmo a emplear, mejora algunos de los ya existentes como el *BallString* [Narváez et al., 2000, 2001], permitiendo que funcione cuando los costes de los enlaces crecen y cuando existen varios cambios; el *DynamicSWSF-FP* [Ramalingan and Reps, 1996], incorporando una fase de mantenimiento del árbol puesto que solo calculaba las distancias de los caminos óptimos; y el Dijkstra de [Narváez et al., 2000], proponiendo una versión que es capaz de trabajar bajo el tipo de dinamismo que ellos tienen en cuenta. Con el fin de demostrar el funcionamiento del algoritmo propuesto, los autores no aportan información sobre su rendimiento en grafos vastos, aunque sí en grafos de hasta 15.000 nodos comprobando que el tiempo que tarda su propuesta en recuperarse de cambios que afectan a un máximo del 5% de los enlaces es mejor que el de algoritmos anteriores.

Otro trabajo que ha adaptado su algoritmo a grafos dinámicos es [Schultes and Sanders, 2007], en el que exploran una variante de las técnicas de indexación jerárquica diseñada para soportar los cambios dinámicos en los costes de los enlaces o en las funciones de coste. Sin embargo, esta aproximación dinámica requiere o bien una reconstrucción del índice cuando los costes (o las funciones de coste) del grafo cambian, o bien que el algoritmo de búsqueda restrinja de manera incremental la utilización de la información del índice creado. Su solución se adapta para funcionar excepcionalmente bien en redes de carreteras, donde los grafos son prácticamente planos y el grado de los nodos es bajo, aunque el preprocesado requiere mucho tiempo y espacio.

En lo referente a autores de los que no se hacía mención en el apartado de estática, se puede encontrar a [Nannicini et al., 2010], en el que no se abordan los grafos estáticos, pero sí la búsqueda en grafos vastos dinámicos. Ya en un trabajo anterior [Nannicini and Liberti, 2008] proponen optimizar los caminos sobre grafos en los que cambian los costes de los enlaces, mediante modificaciones en la creación de los árboles que se emplean para hacer más rápidas

las búsquedas y en los algoritmos utilizados (Dijkstra, A^*, etc.), pero es en [Nannicini et al., 2010] donde se propone un nuevo algoritmo centrado en grafos que representan redes de carreteras en los que el tiempo en recorrer un enlace es almacenado en los costes de cada enlace y cuyas características pueden resumirse en la aplicación de las siguientes técnicas:

- Una jerarquía: para hacerlo se considera al grafo como estático (los costes de cada enlace representan a la media de todos los posibles valores que puede tomar). Esto se hace como preprocesado y solo se recalcula cuando cambia la topología del grafo.
- Propagación de la información dinámica del tráfico: cada minuto se actualiza el tiempo de viaje asignado a cada enlace y la variación se propaga a los enlaces adyacentes teniendo en cuenta su importancia (por ejemplo cuantos carriles tiene) y a algunos un poco más lejanos ponderando los tiempos a actualizar. Hacer esto lleva pocos segundos, de manera que se puede considerar que el algoritmo es adaptable a los cambios que puedan aparecer.
- Petición de camino: se aplica el algoritmo de Dijkstra bidireccional con múltiples niveles operando sobre la jerarquía preprocesada y utilizando los costes dinámicos. La condición de parada viene dada cuando la búsqueda en un sentido alcanza a la del otro sentido. Es una búsqueda muy rápida porque transcurre en su mayoría en el nivel superior (que es el que menos nodos y enlaces tiene). En la experimentación realizada emplea un tiempo de respuesta inferior a un segundo utilizando un ordenador con un multiprocesador Intel Xeon 2.6 GHz con 8GB RAM y disponiendo únicamente de dos niveles.

Otro ejemplo de búsqueda de caminos en grafos vastos dinámicos que representan redes de carreteras es el presentado por Kim et al. [Kim et al., 2007]. En este caso se tiene como fin crear planes de evacuación ante emergencias (calcular el camino óptimo para ir de los inicios a los destinos), y para probar su propuesta emplean grafos de tan solo unos cuantos cientos de nodos en el que los costes de los enlaces representan el nivel máximo de saturación y el tiempo que se tarda en recorrerlos, y el de los nodos únicamente el nivel de saturación. No obstante, en trabajos futuros pretenden aplicar su propuesta sobre grafos vastos almacenados en una base de datos.

Para crear dichos planes de evacuación, calculan previamente la saturación de cada tramo: se sabe el número de usuarios que se tiene y la saturación máxima permitida por enlace, de manera que se puede crear un plan en el que se van ocupando los distintos enlaces, sabiendo que si en el momento *t* el enlace *x* está saturado, en el *t+1* dicho enlace estará libre y podrá volverse a mandar gente por él. Para hacer esto se proponen dos alternativas: Emplear la heurística *ILR* (*Intelligent Load Reduction*), que sabiendo la carga y la saturación propone una planificación completa consiguiendo una mejora del rendimiento a costa de una disminución en la calidad; o

utilizar *IDR* (*Incremental Data Structure*) que se aplica sobre la heurística *CCRP* (*Capacity Constrained Route Planner*) y que la hace escalable gracias a evitar los cálculos redundantes propios del *CCRP*.

2.2.2.2 Almacenamiento en Memoria Secundaria

Con el fin de superar las restricciones de espacio, pero teniendo en cuenta el dinamismo, existe otro grupo de trabajos que presentan propuestas de algoritmos con grafos, y preprocesado, almacenando toda la información que sea necesaria en soporte secundario gestionado a nivel lógico por bases de datos o sistemas de ficheros.

El primero de los trabajos que merece la pena destacar es [Chan and Zhang, 2009], que ha descrito toda una gama de algoritmos adaptados a diferentes conjuntos de requisitos a lo largo del tiempo hasta llegar a este momento. En este trabajo muestra un algoritmo capaz de encontrar caminos entre nodos dentro de grafos vastos dinámicos y que se adaptan rápidamente a los cambios. Se debe destacar que solo consideran cambios en los costes de los enlaces, descartando cambios en los costes de los nodos.

Para ello, realizan una serie de cambios en el preprocesado de los grafos, ya que dicha fase de su propuesta es lo único que dicen se ve afectado por los cambios en los costes de los enlaces. De este modo, el preprocesado consiste en dividir en fragmentos el grafo, guardar los caminos óptimos entre los nodos que pertenecen a varias fronteras a la vez (que denominan *supernodos*), y cuando hay un *supernodo* que se ve afectado por un cambio, se relajan los enlaces del fragmento al que pertenece (se disminuye la probabilidad de que sean elegidos en un camino) si el cambio afecta al fragmento, o en caso de no estar afectado se relajan los enlaces del camino entre *supernodos*. Con este proceso consiguen que aquellos enlaces relajados sean elegidos con una menor probabilidad, impidiendo que se elijan caminos que no sean en ese momento los óptimos.

En lo referente a propuestas realizadas por autores que hasta este momento no se habían mencionado, es interesante destacar los siguientes:

- [Ding et al., 2008]: utiliza Dijkstra para buscar caminos entre nodos de grafos de 16.326 nodos y 26.528 enlaces almacenados en una base de datos teniendo en cuenta que el coste de cada enlace varía con el tiempo. Se basan en calcular el camino óptimo teniendo en cuenta los costes de los enlaces en el momento de llegada, ya que tienen guardados los costes para los distintos tiempos en los distintos enlaces.
- La propuesta presentada en [George et al., 2007], con una base de datos para almacenar grafos que no alcanzan la consideración de *vastos* (la experimentación la realizan con grafos de hasta 786 nodos y 2.106 enlaces). Para representar la variación del grafo con el tiempo utilizan lo que ellos llaman el *time-aggregated graph*, que en lugar de replicar el

grafo cada instante de tiempo, solo guarda información de aquellos enlaces que han cambiado (variando su coste o desapareciendo, ya que solo tienen en cuenta ese tipo de dinamismo), reduciendo de este modo el espacio necesario para almacenar la información. En cuanto a los algoritmos para buscar el camino más corto, utilizan el *BEST* y el *SP-TAG*. El primero informa acerca de cuál pueda ser el mejor momento para partir desde un nodo y llegar al otro teniendo en cuenta que este momento permite que el recorrido se haga en el menor tiempo posible. El *SP-TAG* [George and Shekhar, 2006] busca el camino óptimo partiendo de un tiempo de inicio fijo. Para ambos algoritmos se tiene en cuenta que ya se saben de antemano todos los cambios que va a poder experimentar el grafo, y en qué momento se producen.

Junto a estos algoritmos que intentan buscar el camino óptimo sobre grafos dinámicos, existen otros que se centran en que el preprocesamiento realizado se mantenga consistente a pesar de los cambios que puedan producirse (tal y como hacía [Chan and Yang, 2009] en el caso del almacenamiento en memoria principal). Una amplia relación de estos trabajos en el estado del arte puede consultarse en [Yu and Cheng, 2010]. Entre ellos, cabe destacar el de Schenkel et al. [Schenkel et al., 2005] y el de Bramandia et al. [Bramandia et al., 2008, 2009]. El primero vuelve a etiquetar todos los nodos afectados por el cambio de un nodo del grafo, lo que en realidad se aleja de una actualización local y puede suponer un coste elevado. El segundo intenta que el algoritmo sea adaptable a los cambios calculando caminos alternativos a los almacenados (resultantes del preprocesado).

2.2.2.3 Conclusiones

Adoptando la definición de dinamismo indicada en [Nannicini and Liberti, 2008], se dirá que un grafo es dinámico cuando en él se realizan una serie de cambios, pudiendo ser estos variaciones en los costes de los enlaces o nodos, aparición de nodos/enlaces, o desaparición de nodos/enlaces.

No obstante, a pesar de esta definición, tal y como se ha podido observar en el repaso de los distintos trabajos existentes en el área, casi todos ellos se centran únicamente en la variación de los costes de los enlaces (ya sea una variación creciente, decreciente, o ambas) alegando que es el cambio más típico y obviando el resto de posibles variaciones. Algunos justifican su elección de tipo de dinamismo indicando que la desaparición de enlaces se puede representar como una variación del coste del mismo donde pasa a tomar un valor infinito, pero la aparición de nuevos enlaces/nodos no existentes previamente es imposible de reflejar mediante variaciones de costes a menos que esos elementos hayan sido previstos e incluidos de inicio con un coste infinito, que en un momento dado es reducido hasta un valor que lo hace practicable.

Además de este problema, se encuentra el de que en la mayoría de los casos la adaptación al dinamismo se refiere a procesos de actualización (globales o locales) orientados a mantener consistente el conocimiento obtenido en el preproceso. Sin embargo, frecuentemente se omiten las siguientes cuestiones:

- Si las variaciones son frecuentes, o si las mismas afectan a diversas zonas del preprocesado (árbol, fragmentación con cálculos de caminos, etc.), es imposible obtener un camino en tiempo porque se tendría que esperar al momento en el cual no haya cambios y el preprocesado se mantenga estable, pudiendo ocurrir que esto no pasase en ningún momento y que nunca se llegase a poder dar una solución.
- Los cambios pueden afectar al algoritmo de búsqueda en sí, de manera que el camino encontrado no sería válido y la búsqueda tendría que empezar desde el principio, lo que implicaría un aumento en el tiempo de respuesta dándose el caso de que nunca se mostrase una solución por haber sobrepasado el límite de espera permitido.

Es por todo ello que surge la necesidad de encontrar un algoritmo de búsqueda al que no afecten los cambios del grafo mientras esté resolviéndose, es decir, que sea capaz de adaptarse rápidamente a esos cambios, y que además tenga un preprocesado (para poder tratar con el tamaño de los grafos) de cómputo razonablemente ligero y que no afecte al algoritmo de búsqueda cuando éste se encuentra en ejecución. Es decir, que uno ayudase al otro, pero que no se interfiriesen de manera negativa.

Además de las conclusiones relativas al funcionamiento de las propuestas que se enfrentan al dinamismo, es interesante señalar que casi todos los trabajos sobre grafos vastos restringen su dominio de aplicación a redes de carreteras, justificando su elección en el hecho de que en este dominio es en el que más frecuentemente se solicitan búsquedas del camino óptimo. No obstante, este es un dominio muy limitado y existen otras topologías de grafo sobre las que cualquier avance tendría gran impacto en el estado del arte, ya que su utilización es ya bastante extendida o podría llegar a serlo si a estos grafos se les dotara de las herramientas adecuadas. Éstas son, por ejemplo, las redes sociales, las redes de interacción de proteínas, o los grafos de la Web. Estos exhiben diferentes grados en los nodos así como características estructurales distintas a las de las redes de carreteras (que son en su mayoría grafos planos), y contienen cientos de millones, o incluso billones, de nodos. Además, en casos como el de las redes sociales, dichos grafos presentan características dinámicas (con cambios frecuentes), ya que se apuntan nuevos usuarios, se borran otros, se incluyen nuevas relaciones entre las distintas personas que forman la red social, etc.

Por el creciente interés de este tipo de grafos, así como por la carencia de algoritmos de búsqueda de caminos existentes sobre ellos y sus características distintas a las típicas

encontradas en aplicaciones más habituales (como las famosas redes de carreteras), en el siguiente apartado van a estudiarse en detalle estos grafos que se engloban bajo la denominación común de *Redes Complejas* (*Complex Networks*).

2.2.3 Redes Complejas

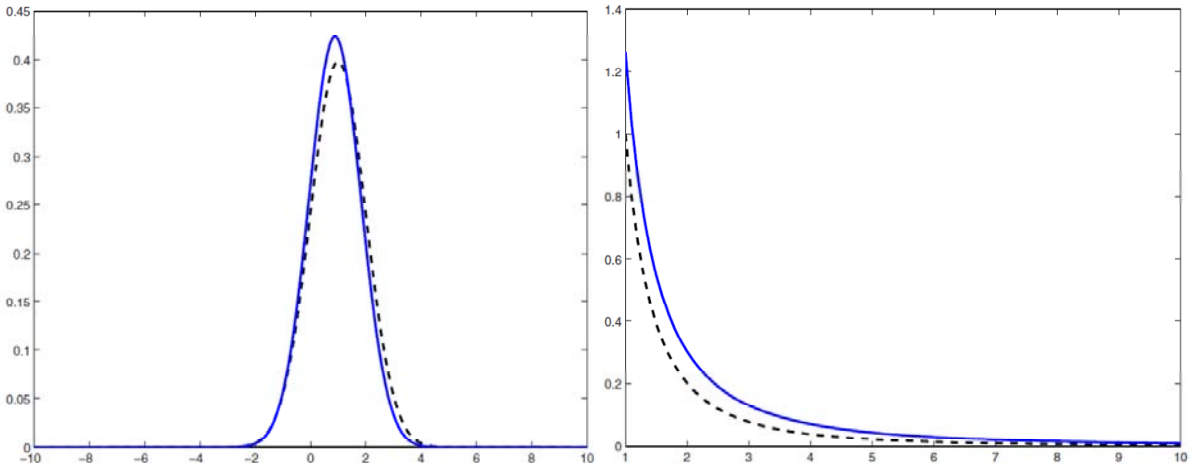
En la actualidad existen gran cantidad de aplicaciones en las que los grafos que las representan se enmarcan dentro de lo que se conoce como *Redes Complejas* (*Complex Networks*). Aunque, tal y como queda demostrado en [Newman et al., 2006], este tipo de grafos puede encontrarse en redes biológicas, redes neuronales o redes de citación de artículos [Tang et al., 2008], el interés en ellas ha crecido desde la aparición de la Web 2.0 e Internet por su alta utilización, motivada principalmente por las redes sociales y las aplicaciones sociales que permiten intercambiar conocimiento entre distintas personas (redes de comunicaciones de email [Diesner et al., 2005], redes de mensajería instantánea [Leskovec and Horvitz, 2008], redes de telefonía móvil [Nanavati et al., 2006], o redes de amigos [Mislove et al., 2007]) ya que todas estas aplicaciones las emplean.

La característica principal de este tipo de redes es que tienen un tamaño extenso, de manera que propician la aparición de estructuras que no son observables en redes más pequeñas, planteando gran cantidad de retos.

La mayoría de las redes complejas poseen una serie de características en común que las hace interesantes y distintas con respecto al resto de grafos, y sobre todo, con respecto a grafos de pequeño tamaño. Entre estas características, las más significativas son las siguientes:

- Distribución libre de escala (*Scale free distribution*): en las redes complejas, el grado de los nodos no sigue una distribución normal, sino que sigue una distribución de ley de potencias. Es decir, mientras que en la mayoría de los grafos existe un grado de nodo que poseen la mayoría de los nodos, y todos los demás se encuentran en torno a él Figura 2.1 (a), en las redes complejas existen pocos nodos con un grado muy alto y el resto lo tiene bajo Figura 2.1 (b). A las redes que poseen este tipo de distribución de grados se las conoce como *Redes de Escala Libre* (*Scale-Free Network*).
- Efecto del mundo pequeño (*Small-World Effect*): en 1969 Milgram y Travers [Travers and Milgram, 1969] realizaron un experimento en Estados Unidos que demostraba que cualquier persona podía alcanzar a cualquier otra en seis pasos. Este efecto se ha observado en distintas redes de gran tamaño y ha quedado demostrado en distintos trabajos. Por ejemplo, en [Watts et al., 2002] se crea un modelo que explica el porqué de que sea sencillo y rápido encontrar el nodo destino, ya esté cerca o lejos del inicio, dentro de una red compleja disponiendo únicamente de la información local.

Es decir, en las redes complejas, el número de pasos que componen el camino óptimo es pequeño con respecto al tamaño de la red. Las redes que cumplen esta característica se denominan *Redes de Mundo Pequeño* (*Small-World Networks* [Watts and Strogatz, 1998]).



(a) Distribución normal (b) Distribucion por ley de potencias

Figura 2.1 [Tang and Liu, 2010]. Diferentes distribuciones del grado de los nodos del grafo. La línea discontinua es una distribución original y la azul una estimación basada en 100 muestras obtenidas de una distribución real.

- Altos valores del coeficiente de agrupamiento (*clustering coefficient*). Entendiendo por coeficiente de agrupamiento cuánto de agrupado o interconectado está un nodo con el resto de nodos del grafo, se da el caso de que en las redes complejas existe un gran número de nodos que poseen un alto valor para dicho coeficiente, es decir, que están conectados con una gran cantidad de nodos del grafo.

Para ampliar la información de cada una de estas características puede consultarse [Albert and Barabási, 2002; Barabási and Bonabeau, 2003; Boccaletti et al., 2006; Dorogovtsev and Mendes, 2004; Newman, 2003; Watts, 2004].

Aunque estrictamente el término *Small-World Networks* hace referencia a las redes que cumplen únicamente el segundo requisito, en la actualidad la aplicación de dicho término se ha extendido y hace referencia a aquellos grafos que satisfacen las tres características. Por lo tanto, cuando en el resto del documento se hable de una *Small-World Network*, se entenderá un grafo que cumple que tiene un alto valor de coeficiente de agrupamiento, que sigue una distribución libre de escala, y que posee el efecto del mundo pequeño.

Una vez establecidas las características de este tipo de grafos, es importante indicar que existen gran cantidad de trabajos enfocados a dar solución a los distintos retos que plantean. Por ejemplo, en el caso de las redes sociales, uno de los primeros problemas que se encuentra a la hora de tratar con ellas es modelarlas, es decir, decidir que característica es la que se tendrá en cuenta para ponderar los costes de los enlaces. Aunque esto podría resultar sencillo, no es una tarea fácil. Por ejemplo, en [Adamic and Adar, 2005] se observa esta problemática al organizar

una red social de amigos (una red social llamada *Club Nexus* de la Universidad de Stanford). La dificultad de establecer los costes viene determinada por el desconocimiento de algunos datos y porque los criterios para organizar la información no son claros (se puede tener en cuenta la edad, el género, el año de promoción, etc.).

No obstante, siguiendo en el caso de las redes sociales, una vez se tiene resuelto el problema del modelado, queda realizar sobre estos grafos todas las tareas que se llevan a cabo en cualquiera de los otros grafos existentes. Por ejemplo, resolver el problema de los cuatro colores [Walsh, 1999], o contar con simuladores que permitan crear redes con las mismas características que las reales como [Zhang et al., 2007] en el que se propone un simulador de redes de expertos, de manera que se crean grafos con las mismas características que redes de expertos reales on-line como *Java Forum*, incorporando además algoritmos para buscar las personas más expertas en distintos temas.

Sin embargo, como se indica en [Kautz et al., 1997], dentro del dominio de las redes sociales una de las peticiones más solicitadas es la de encontrar un miembro concreto dentro de la red para conocer si existe algún tipo de relación con el miembro que realiza la petición, ya que esto favorece las alianzas de negocios o de investigación entre personas que, de otra manera, habrían tenido dificultades para hablar o conocerse. En otras palabras, la consulta más frecuente que se realiza dentro de una red social es la de la búsqueda del camino que une dos puntos. Una de las primeras propuestas para resolver la búsqueda de miembros de una red apareció en 1997 [Kautz et al., 1997], presentando un sistema (*ReferralWeb*) que realizaba búsquedas en redes sociales de pequeño tamaño: buscaba obtener la cadena de personas para llegar de una persona a otra, es decir, encontrar lo que ellos denominaban *referral chain*.

Este interés en buscar caminos dentro de las redes sociales se puede ampliar a prácticamente cualquier grafo que satisfaga las condiciones para ser un *Small-World*, ya que es interesante buscar algún algoritmo que, aprovechando las características propias de esta topología, proporcione una solución en poco tiempo salvando la dificultad de su gran tamaño. Prueba de este interés es la gran cantidad de trabajos que existen enfocados a la búsqueda de documentos dentro de las redes *Peer-to-Peer (P2P)*, en las que los nodos indican documentos del sistema y los enlaces la relación existente entre cada uno de dichos documentos.

Por ejemplo, en [Li et al., 2004b], para facilitar la búsqueda de caminos, a cada nodo le asignan una tabla que, por cada nodo alcanzable desde el nodo actual, indica una palabra clave que muestra lo que contiene, cuantos pasos hay que dar para llegar a él, y el nodo adyacente a utilizar para alcanzarlo. En caso de buscar por varias palabras clave, se mira si se tiene la información localmente, y si no es así, se hace una tabla por cada vecino y se elige el más cercano.

Otra alternativa para llevar a cabo la búsqueda de documentos es la presentada por [Li et al., 2004a, 2008]. En este caso la red se divide en grupos de documentos según las similitudes entre ellos y se crea un árbol binario teniendo en cuenta dichas similitudes para relacionar unos grupos con otros. Sobre estas organizaciones es donde se realiza la búsqueda de documentos (uno o varios pues también responden consultas de rango) observando si desde donde se parte ya se está en el grupo final o si se tiene que navegar por el árbol. Para demostrar el funcionamiento de su algoritmo emplean grafos de pequeño tamaño (16.384 nodos máximo), no siendo recomendable su aplicación a grafos de mayor tamaño.

Como último ejemplo dentro de las redes *Peer-to-Peer* se encuentra la propuesta de [Jin et al., 2006], en la que todos los nodos tienen una serie de palabras clave asociadas que los identifican. Además, todos tienen una lista de nodos cercanos (como mucho a dos saltos de distancia) que son similares a ellos semánticamente hablando, y otros que están más lejos pero que también cumplen unos requisitos de similitud. De esta manera, cuando se busca el camino proponen dos cosas: buscar en paralelo teniendo en cuenta los nodos apuntados en cada nodo (cercanos y lejanos), o en paralelo mandando la petición a todos los vecinos y además a los nodos lejanos apuntados. Aunque indican que de esta manera buscan de modo eficiente, solo prueban con grafos de hasta 6.000 nodos.

En lo que respecta a la búsqueda en redes sociales, cabe destacar los siguientes trabajos:

- [Kleinberg, 2000a, 2000b]: indica que es posible realizar una búsqueda descentralizada de los caminos teniendo únicamente información local, es decir, cada usuario solo conoce información acerca de sus *K* vecinos más cercanos y tiene *r* contactos aleatorios dentro del grafo alejados de él. De esta manera, el éxito de la búsqueda de una persona desde otra dentro de una red social viene fijado por la distancia que los separa, de modo que cuanto más lejos estén el uno del otro, más complicado será tener éxito. Esto viene de que para ser aceptado por una persona final, inicialmente se ha tenido que ser aceptado por todas las personas del camino, de modo que en el momento en el que esto no se cumpla en uno de los contactos intermedios, no se podrá alcanzar el destino. Por consiguiente, cuantos más contactos haya en el recorrido, mayor será la probabilidad de fracaso.

  Además de esto, introduce el concepto de *grafo navegable*, de manera que se entenderá como tal a todo grafo en el que la búsqueda del nodo destino se pueda realizar de una manera eficiente, e indica que la distribución de los *r* contactos elegidos para cada nodo, y que se emplean para realizar las búsqueda de una manera más rápida, determinará la navegabilidad del grafo.

- [Newman, 2001]: plantea, entre otras cosas, un algoritmo para buscar el camino entre dos nodos de un grafo que se basa en recorrer el grafo desde el nodo destino e ir asignando a cada nodo adyacente la distancia acumulada hasta llegar a él así como el nodo desde el que alcanzarlo, de manera que, una vez se tiene esto, el camino entre el nodo inicio y el nodo destino se calcularía yendo de nodo en nodo tomando el nodo indicado como previo.

- [Liben-Nowell, 2005]: estudia una red social real de los Estados Unidos para aplicar la misma metodología de búsqueda de caminos que en [Kleinberg, 2000a, 2000b], y concluye que ésta es navegable. No obstante, para ello realiza una serie de ajustes sobre la teoría de Kleinberg para tener en cuenta que la distribución geográfica de la población no es homogénea.

  Algunos trabajos posteriores destinados a mejorar esa asignación de contactos lejanos para hacer más navegable los grafos, así como a adaptarse a otras características y a hacer esta metodología válida en redes reales, se presentan en [Liben-Nowell, 2010] y [Mogren et al., 2009].

- [Sandberg, 2006]: indica cómo llegar de una persona a otra en una red social eligiendo siempre como siguiente nodo el adyacente que menos dista del nodo en el que se está en cada momento. Para ello buscan en distintos grafos obteniendo una tasa de éxito baja, a pesar de utilizar redes de tamaño pequeño (grafo de una dimensión de $1.000\text{-}2^7 \cdot 1.000$ nodos con distintas distribuciones, grafo de dos dimensiones con nodos entre 1.024 y $4^3 \cdot 1.024$, o la red real *Pretty Good Privacy* restringida a 23.000 usuarios para obtener algunas respuestas satisfactorias). Señala que para mejorar los resultados habría que aplicar una jerarquía.

- [Honiden et al., 2010]: además de buscar sobre redes sociales, también trabaja sobre otros tipos, todas ellas representadas con grafos de millones de nodos: redes de interacción de proteínas, redes de transporte, etc., y propone un algoritmo para buscar caminos en grafos vastos apoyándose en los grafos duales de Voronoi [Mehlhorn, 1988; Erwing, 2000]. Para ello, cuenta con una fase de preprocesado en la que crea el grafo dual de Voronoi (cuyo tamaño será menor que el original), de modo que cuando se realiza una consulta primero se busca en el nuevo grafo creado, y utilizando la información obtenida, se busca en el grafo original. Tanto en una búsqueda como en otra utilizan el algoritmo de Dijkstra.

Atendiendo a todos los trabajos presentados, se puede concluir que existen gran cantidad de propuestas que tratan de dar solución a distintos problemas planteados sobre las *Redes*

Complejas, distinguiéndose entre todos ellos, por presentar mayores frecuencias e impacto, el de la búsqueda de caminos para llegar de un nodo a otro.

No obstante, en todos ellos, la búsqueda se realiza apoyándose en una reordenación del grafo original o en información extra añadida al grafo para poder dar una solución en el menor tiempo posible. La desventaja de esto es que, al igual que se indicaba en apartados anteriores, este tipo de algoritmos sólo es válido cuando los grafos sobre los que se trabaja son estáticos. No obstante, esta topología suele aparecer en grafos que representan dominios con variaciones temporales bastante frecuentes. Por ejemplo, en una red social aparecen a cada momento nuevos usuarios, o nuevos enlaces entre usuarios. De manera que los trabajos explicados dejan de tener validez por los mismos motivos que se explicaban en apartados anteriores.

Entre los trabajos que sí tienen en cuenta este requisito se encuentra [Feng et al., 2006], en el cual se presenta el algoritmo *SWS* encargado de organizar el grafo y buscar caminos en sistemas *Peer-to-Peer*. En este algoritmo, la forma de organizar los nodos es mediante una jerarquía en la cual se ponen juntos aquellos que tienen información similar y donde el peso de los enlaces está relacionado con la información que comparten sus extremos, además, todos los nodos tienen información de los nodos locales. A la hora de buscar se mira si con la información que tiene el nodo se alcanza el destino, y de no ser así, se elige el nodo más próximo (el de información más similar). Esto se repite hasta alcanzar el nuevo destino. En cuanto al dinamismo: cuando aparece un nodo se crea su tabla de información local, y sin embargo cuando un nodo se elimina no se borra de los nodos adyacentes hasta que no pasa un tiempo límite para evitar borrar e insertar constantemente.

Otro trabajo que trata de resolver el problema del dinamismo en redes complejas es el presentado en [Madduri and Bader, 2009]. En este trabajo se utilizan grafos de millones de nodos con un gran dinamismo (de media 25 millones de inserciones/borrados de enlaces cada segundo) aplicando cachés y árboles, y distribuyendo los procesos en distintos procesadores. Este trabajo busca responder a la pregunta de la existencia de caminos entre dos nodos, no a la de buscar la longitud del camino que los une. Esta operación es más eficiente gracias a que solo hay que mantener en un estado consistente los árboles.

Después de ver estos dos ejemplos, se puede indicar que el dinamismo tratado se centra más en el mantenimiento del preprocesado que en un algoritmo de búsqueda capaz de afrontar las variaciones en el grafo mientras se está realizando la búsqueda del camino. Es decir, en este tipo de grafos se tienen los mismos problemas que en el caso de las redes de carreteras o redes más comunes, incluyendo, además, otro nuevo que señalan algunos de los trabajos anteriores: los algoritmos presentados disminuyen la tasa de éxito a medida que la distancia entre los nodos inicio/destino del camino crece.

Es por esto que es necesario encontrar un algoritmo capaz de solucionar este hueco que dejan los trabajos existentes hasta la actualidad, y que, además, se adapte a las características especiales de las *Redes Complejas* ya que, tal y como demuestra [Yuan et al., 2010], éstas mantienen su topología de *Small-World* a pesar del dinamismo que puedan experimentar. Dicho estudio fue realizado sobre las *trust networks*, que son redes de confianza como los sistemas de recomendación, o redes de seguridad, en las que los enlaces reflejan la confianza que existe de un nodo al otro (entendiendo por confianza, por ejemplo, cuanto de experto es un usuario según otro en un tema concreto). El hecho de elegir este tipo de redes se basó en que poseen un alto dinamismo, pues cada pocos segundos aparecen/desaparecen nodos y enlaces y/o los costes de estos últimos varían. En concreto se empleó la red *Epinions* (www.epinions.com).

2.3 Algoritmos Basados en Colonias de Hormigas

Hasta ahora se han presentado algoritmos de dos tipos: exhaustivos y heurísticos. Con respecto a los primeros, aunque son algoritmos que garantizan encontrar el óptimo global de cualquier problema poseen el inconveniente de que no son aplicables en grafos vastos debido a que su tiempo de respuesta crece de forma exponencial con el tamaño del problema. En cambio, los algoritmos heurísticos son normalmente bastante rápidos, pero la calidad de las soluciones encontradas está habitualmente lejos de ser óptima. Además, otro inconveniente de los heurísticos es que no son fáciles de definir en determinados problemas.

Con el fin de solucionar todos estos problemas, a finales de los setenta surgió un nuevo tipo de algoritmos que buscaba resolver los problemas de búsquedas dentro de grafos de una manera eficaz y eficiente, y que a partir de [Glover, 1986] se pasaron a denominar *metaheurísticas*.

Estos nuevos algoritmos, no exactos y generalmente no deterministas, se caracterizan por ser métodos de alto nivel de abstracción que aplican una serie de estrategias o plantillas generales para guiar el proceso de búsqueda de manera que tienen como objetivo encontrar soluciones quasi-óptimas en el menor tiempo posible gracias a la incorporación de mecanismos que evitan regiones no prometedoras del espacio de búsqueda. Además, el esquema básico de cualquier metaheurística tiene una estructura predefinida y utiliza el conocimiento del problema que trata de resolver en forma de heurísticos específicos que son controlados por una estrategia de más alto nivel [Chicano, 2007].

Las distintas metaheurísticas se pueden clasificar de diversas maneras teniendo en cuenta las características que las componen [Cantú-Paz, 2000; Crainic and Toulouse, 2003], aunque la más popular es la que las divide en dos grupos: las *basadas en trayectoria* y las *basadas en población*. El primer tipo incluirá a aquellas metaheurísticas que parten de un punto inicial y van actualizando la solución presente mediante la exploración de los nodos adyacentes,

formando una trayectoria. La búsqueda finaliza cuando se alcanza un número máximo de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso [Gómez González, 2008]. En lo referente al segundo tipo, incluirá a aquellos algoritmos que trabajan con un conjunto de soluciones en cada iteración.

Dentro del primer grupo de metaheurísticas se encuentra el Enfriamiento Simulado (*Simulated Anneling*) [Kirkpatrick et al., 1983], que se basa en el proceso de enfriamiento de los metales. En cada iteración se elige una solución *s'* a partir de la solución actual *s*, de manera que si *s'* es mejor que *s*, se sustituye *s* por *s'* pasando a ser *s'* la solución actual. En caso de no ser así (si la solución *s'* es peor que *s*), *s'* es aceptada con una determinada probabilidad. Gracias al hecho de aceptar casos peores que el actual con una determinada probabilidad, se consigue evitar caer en mínimos locales.

Junto a este método está el de la Búsqueda Tabú (*Tabu Search*), cuyos fundamentos fueron introducidos en [Glover, 1986] tomando como base [Glover, 1977]. Este algoritmo consiste en el almacenamiento de las soluciones que se van eligiendo a lo largo de la búsqueda, de manera que cuando se tenga que tomar la decisión de cual solución escoger de entre todas las posibles en cada iteración, se eliminarán de ese conjunto aquellas que ya hayan sido utilizadas anteriormente. Gracias a esta manera de funcionar, se evitan los mínimos locales, así como explorar zonas anteriormente visitadas. Más información de este tipo de algoritmo puede encontrarse en [Glover and Laguna, 1997].

Un algoritmo muy general y con muchos grados de libertad que se engloba dentro de las metaheurísticas basadas en trayectorias es el de la Búsqueda de Vecinos Variable (*Variable Neighborhood Search*) [Mladenovic and Hansen, 1997]. En él, en una primera fase de inicialización se definen una serie de vecinos basándose en una elección aleatoria o en base a unas ecuaciones más complejas. Una vez se ha hecho esto, en cada iteración se tienen tres fases: la elección del candidato en la que se elige aleatoriamente un vecino *s'* de *s* utilizando el *k-ésimo* vecino; la fase de mejora en la que se intenta mejorar *s'* examinando los vecinos, de manera que si la solución se mejora en la primera búsqueda se asigna a *k* el valor uno (*k=1*), y si no es así se prueba con *k=k+1*, y así sucesivamente; y el movimiento a la solución elegida.

Por último dentro de este grupo destaca también el método de Búsqueda Local Iterada (*Iterated Local Search*) [Lourenço et al., 2002; Stützle, 1999] en el que en cada iteración, la solución actual es perturbada para aplicarle posteriormente un método de búsqueda local para mejorarla. El valor obtenido será aceptado si pasa un test de aceptación.

En cuanto a las metaheurísticas basadas en población, cabe destacar las siguientes:

- Algoritmos Evolutivos (*Evolutionary Algorithms*) [Bäck et al., 1997]: algoritmo introducido en 1970 por John Holland [Holland, 1975], es un método que trata de

encontrar una solución inspirándose en la evolución biológica y en la selección natural. Dicho algoritmo consiste en la selección inicial de una población aleatoria y en una sucesión de iteraciones posteriores hasta llegar a un cierto criterio de finalización. En cada una de las iteraciones se realizan tres pasos: selección de los individuos más aptos, reproducción de los individuos seleccionados mediante una combinación entre ellos para dar origen a nuevos individuos sobre los que aplicar mutaciones, y reemplazo, que crea una nueva población a partir de la población actual y/o los mejores nuevos individuos generados.

- Algoritmos de Estimación de la Distribución (*Estimation of Distribution Algorithms*) [Mühlenbein, 1998]: son algoritmos muy similares a los anteriores, pero en lugar de aplicar las funciones de reproducción y mutación para crear una nueva población, lo que se hace es emplear la distribución de probabilidad calculada a partir de los mejores individuos de la población anterior.

- Búsqueda Dispersa (*Scatter Search*) [Glober, 1977, 1998; Glober and Kochenberger, 2002]: este algoritmo crea un conjunto de soluciones posibles con una buena calidad y otras distribuidas por todo el espacio de búsqueda. A este conjunto de soluciones se le denomina conjunto de referencia, y una vez obtenido, se combinarán las soluciones que contiene con el fin de alcanzar otro nuevo conjunto mejorado hasta satisfacer una determinada condición de terminación.

- Algoritmos de Optimización Basados en Colonias de Hormigas (*Ant Colony Optimization (ACO)* [Dorigo, 1992; Dorigo and Stützle, 2003]): simulan el comportamiento de las hormigas reales: todas las hormigas parten del hormiguero y tratan de buscar comida, de manera que cuando la encuentran la llevan al hormiguero. Durante este proceso, las hormigas depositan una sustancia química denominada *feromona* que permite a las otras hormigas del hormiguero saber si sus congéneres han pasado por ese punto y encontrar por tanto, de una manera rápida, el camino a la comida. Es este rastro de feromona lo que permite que el camino que se encuentra sea el óptimo (hecho que quedó demostrado por el experimento llevado a cabo en [Goss et al., 1989]). La forma de simularlo con las hormigas artificiales es mediante la aplicación de un modelo probabilístico.

- Algoritmos de Optimización Basados en Cúmulos de Partículas (*Particle Swarm Optimization* [Kennedy, 1997, 1999; Kennedy and Eberhart, 1995, 1997; Kennedy et al., 2001]): son algoritmos que se basan en el comportamiento de vuelo de las bandadas o de movimiento de los bancos de peces. Cada elemento del conjunto (*cúmulo*) posee una posición y velocidad que cambia conforme avanza la búsqueda. Los parámetros que

afectan al cambio en el movimiento de cada partícula son la velocidad y las posiciones en las que ella y las partículas de su *vecindario* (un conjunto de partículas del cúmulo) encontraron buenas soluciones. En cuanto al vecindario de una partícula, éste puede ser *global* (todas las partículas del cúmulo se consideran vecinas) o *local* (solo las partículas más cercanas se consideran vecinas).

Aunque todas estas metaheurísticas resuelven problemas encontrados en los algoritmos heurísticos y en los exhaustivos, es el algoritmo de optimización basado en colonias de hormigas el que mejor se adapta al problema de búsqueda de caminos dentro de grafos, incluso cuando estos tienen un comportamiento dinámico, gracias al modelo probabilístico en el que se apoya para seleccionar el siguiente nodo que visitar. Esta adaptabilidad queda demostrada en trabajos como, por ejemplo, [Kadono et al., 2010], en el que se emplea este tipo de algoritmos para la búsqueda del camino más corto dentro de redes ad hoc.

2.3.1 Características Principales y Variantes de ACO

Los algoritmos basados en colonias de hormigas son métodos que tratan de imitar el comportamiento que poseen las hormigas en la naturaleza, ya que dicho comportamiento permite encontrar el camino de menor longitud entre el hormiguero y la comida, lo que traducido a grafos implicaría encontrar el camino más corto entre cualquier par de nodos.

Las hormigas en la naturaleza actúan como elementos simples que forman un conjunto coordinado capaz de encontrar una fuente de comida de una manera rápida y empleando un bajo coste de camino dentro de una determinada superficie. Esto lo pueden realizar gracias a un tipo de comunicación indirecta que permite ayudarse unas a otras. Este tipo de comunicación se rige por el seguimiento del rastro de una sustancia química que depositan a lo largo del camino seguido y que se denomina *feromona*.

En las hormigas artificiales se imita este comportamiento, de manera que lo que se hace es que cada hormiga (agente dentro del sistema) se coordina con el resto para resolver problemas de optimización comunicándose unas con otras simulando el rastro de feromonas mediante una variable y utilizándolo para determinar el siguiente paso a dar por medio de una serie de funciones probabilísticas. De esta manera son capaces de encontrar un camino suficientemente cercano a la solución óptima.

En cuanto a la representación del problema sobre el que trabajar, lo que se hace es utilizar grafos, de manera que el movimiento de las hormigas se limita a ir de un nodo a otro utilizando los enlaces que los unen, y depositando la feromona en los enlaces (procedimiento habitual) o en los nodos (variante útil cuando el orden de los nodos que se recorren en el camino no es relevante, es decir, en problemas de subconjunto [Leguizamón and Michalewicz, 1999]).

La primera propuesta de simulación del comportamiento de las colonias de hormigas se denominó *Ant System* y fue realizada por Dorigo et al. [Dorigo et al., 1991, 1996]. Dicha propuesta trataba de resolver el problema clásico de optimización del problema del viajante (*Travel Salesman Problem*) sobre grafos pequeños (menos de cien nodos), aunque también mostraba como adaptarlo a otro tipo de problemas. A pesar de que los resultados obtenidos no fueron del todo satisfactorios por su baja eficiencia, despertó un gran interés y se realizaron multitud de trabajos posteriores tratando de mejorarlo.

2.3.1.1 Sistema de Hormigas o Algoritmo Básico (Ant System)

Este fue el primer algoritmo basado en el comportamiento de las colonias de hormigas con el fin de encontrar el camino óptimo entre dos puntos.

En concreto, lo que se hace es representar el problema a tratar mediante un grafo G en el cual hay una serie de nodos N y enlaces L, formalmente, $G = \{N, L\}$ tal que $N = \{i\}$ y $L = \{l_{ij}\}$, y las hormigas parten de nodos de ese grafo (nodos inicio que simulan hormigueros) tratando de buscar otro nodo (nodo destino).

Con el fin de que las hormigas se muevan aleatoriamente por G teniendo en cuenta los movimientos que hayan realizado otras compañeras de la colonia, cada uno de los enlaces $l_{ij} \in L$ va a tener asociado un rastro de feromona τ_{ij}, y es posible que también se tenga un valor heurístico η_{ij}. Tal y como se ha explicado anteriormente, también podría darse el caso de que la feromona fuese depositada en los nodos. No obstante, puesto que en la búsqueda de caminos el orden en el que se deben recorrer los nodos sí que es relevante, este tipo de depósito no se tendrá en cuenta. Es importante indicar, que mientras que el rastro de feromona se modificará a lo largo de la ejecución, el valor heurístico se mantendrá constante (es un valor impuesto por el analista).

Una vez se tiene definido el entorno en el cual se realizará la búsqueda, se pasan a describir cada una de las fases que lo forman, y cuya combinación permite que el camino encontrado, cuando hayan terminado todas las iteraciones, sea el óptimo:

2.3.1.1.1 Fase de Construcción

Durante esta fase las hormigas se mueven por el grafo yendo de un nodo a otro hasta completar un camino.

La elección del siguiente nodo al que moverse se hace en función de la probabilidad mostrada en la Ecuación 2.1. En ella, por cada hormiga *a* situada en el nodo *i*, se muestra la probabilidad de elegir como siguiente nodo cada uno de nodos adyacentes y que se registran en el conjunto *Adj(i)*. Es decir, muestra la probabilidad de elegir cada uno de los nodos *j* dentro del conjunto *Adj(i)*.

$$p_{ij}^{a} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{x \in Adj(i)}[\tau_{ix}]^{\alpha}[\eta_{ix}]^{\beta}} \quad \text{donde } j \in Adj(i)$$

Ecuación 2.1. Probabilidad de elección.

Para ello, se tiene en cuenta la cantidad de feromona depositada en el enlace que une ambos nodos τ_{ij} y la heurística asociada a dicho enlace η_{ij}. Como se puede observar, la influencia de dichas variables viene determinada por los valores de α y β, de manera que si $\alpha = 0$, no existe ninguna comunicación entre las hormigas, y si $\beta = 0$ solo se tiene en cuenta la feromona. Esto puede resultar peligroso y conducir a mínimos locales.

En Figura 2.2 se puede observar un ejemplo de elección del siguiente nodo dentro de esta fase. En dicho ejemplo, la hormiga *a* se encuentra en el nodo *i* y podría elegir ir a cualquiera de los nodos dentro del conjunto *Adj(i) = {j, n, m}* (son los nodos adyacentes al nodo *i*), de tal manera que la probabilidad de ir a alguno de estos nodos sería Ecuación 2.2 (a) para el caso del nodo *j*, Ecuación 2.2 (b) para el caso del nodo *n*, y Ecuación 2.2 (c) para el caso del nodo *m*.

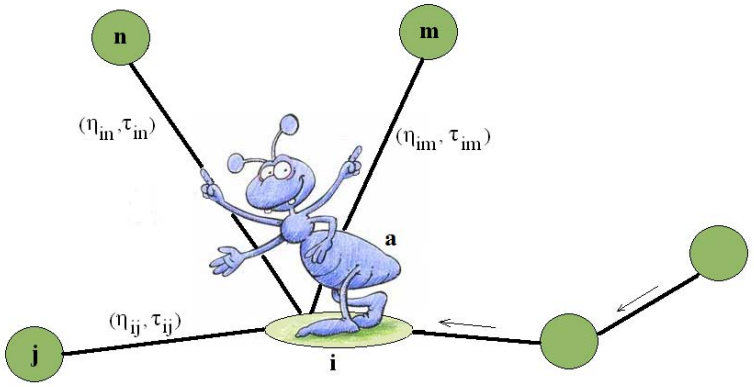


Figura 2.2. Ejemplo de elección de siguiente nodo en la fase de construcción.

Cuando un nodo es elegido como siguiente paso, éste es guardado en una lista, denominada *lista tabú*, que evita volver a repetir ese nodo y que hace posible la reconstrucción del camino seguido una vez finalizada la fase de construcción, pues todos los nodos empleados están almacenados en ella.

$$p_{ij}^{a} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta} + [\tau_{in}]^{\alpha}[\eta_{in}]^{\beta} + [\tau_{im}]^{\alpha}[\eta_{im}]^{\beta}} \quad \text{(a)}$$

$$p_{in}^{a} = \frac{[\tau_{in}]^{\alpha}[\eta_{in}]^{\beta}}{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta} + [\tau_{in}]^{\alpha}[\eta_{in}]^{\beta} + [\tau_{im}]^{\alpha}[\eta_{im}]^{\beta}} \quad \text{(b)}$$

$$p_{im}^{a} = \frac{[\tau_{im}]^{\alpha}[\eta_{im}]^{\beta}}{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta} + [\tau_{in}]^{\alpha}[\eta_{in}]^{\beta} + [\tau_{im}]^{\alpha}[\eta_{im}]^{\beta}} \quad \text{(c)}$$

Ecuación 2.2. Probabilidad de los nodos del siguiente paso.

2.3.1.1.2 Fase de Actualización de Feromona

Al finalizar la fase de construcción (de búsqueda de una solución) se procede a actualizar el rastro de feromona que se encuentra depositado en el grafo.

Este proceso de actualización consta de dos partes, siendo la primera de ellas la *evaporación* de parte de la feromona depositada en los enlaces (con el fin de simular la disipación de dicho rastro que se produce en la naturaleza) para evitar la rápida convergencia hacia mínimos locales. La forma de hacerlo es aplicando la Ecuación 2.3 en todos los enlaces del grafo. En dicha ecuación, τ_{ij} es la feromona depositada en el enlace l_{ij} y ρ representa al *factor de evaporación* que toma valores dentro del intervalo *(0, 1]*.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij}$$

Ecuación 2.3. Evaporación de la feromona.

Una vez se ha producido la evaporación de parte de la feromona se pasa a la fase de *depósito* de feromona, es decir, se incrementa el valor de la feromona en aquellos enlaces que pertenecen a la ruta encontrada. Para ello, se sigue la Ecuación 2.4, en la que τ_{ij} es la feromona depositada en el enlace l_{ij} y $\Delta\tau_{ij}^{a}$ es el factor con el que se va a incrementar la feromona depositada en el enlace l_{ij}. Dicho factor va a depender del coste del camino encontrado (suele ser inversamente proporcional a la longitud del camino) por la hormiga *a*.

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^{a} \quad , \forall l_{ij} \in \text{Camino}$$

Ecuación 2.4. Depósito de feromona.

A este tipo de actualización de feromona se le conoce habitualmente con el nombre de *Actualización Global*, pues se realiza cuando ya se ha finalizado la búsqueda y se aplica sobre todo el camino encontrado.

La unión de estas dos fases dota a los algoritmos basados en colonias de hormigas de una rápida adaptación al entorno, y es capaz de encontrar el camino óptimo en caso de existir distintas alternativas para ir desde el nodo inicio (hormiguero) al nodo destino (la comida) tal y como ocurre en la naturaleza.

Por ejemplo, en la Figura 2.3 (a) se desea ir del hormiguero (situado en la parte baja del dibujo) al sándwich. Para ello, existen dos alternativas. De ellas, tal y como se puede apreciar en Figura 2.3 (a), la de la derecha es más larga que la de la izquierda.

Cuando el algoritmo empieza la búsqueda, debido a que no existe rastro de feromona alguno, los dos caminos tienen la misma probabilidad de ser elegidos, y tal y como se observa en la Figura 2.3 (a), las hormigas van unas por un lado y otras por otro.

Al transcurrir un tiempo, véase Figura 2.3 (b), las hormigas que iban por el camino corto alcanzarán el destino (pues tardan menos en completar el trayecto) mientras que las que

eligieron la otra alternativa todavía no habrán llegado al sándwich. Esto implicará que la feromona que depositarán en ese camino será mayor que la del otro, pues realizan la *actualización global*. De esta forma, según va pasando el tiempo (Figura 2.3 (c) - Figura 2.3 (e)) el número de hormigas que utilizarán el camino corto será mayor que el del camino largo, pues serán atraídas por una cantidad cada vez mayor de feromona que se encuentra en dicho camino, y que cada vez difiere más de la cantidad de feromona del camino largo.

No obstante, tal y como se aprecia en Figura 2.3 (e), debido a que la elección del siguiente paso a dar se basa en una función probabilística, siempre existirá alguna hormiga que elija el otro camino, dejando la puerta abierta al cambio para, por ejemplo, encontrar un camino alternativo en caso de existir alguna alteración en el escenario, y evitando así los mínimos locales.

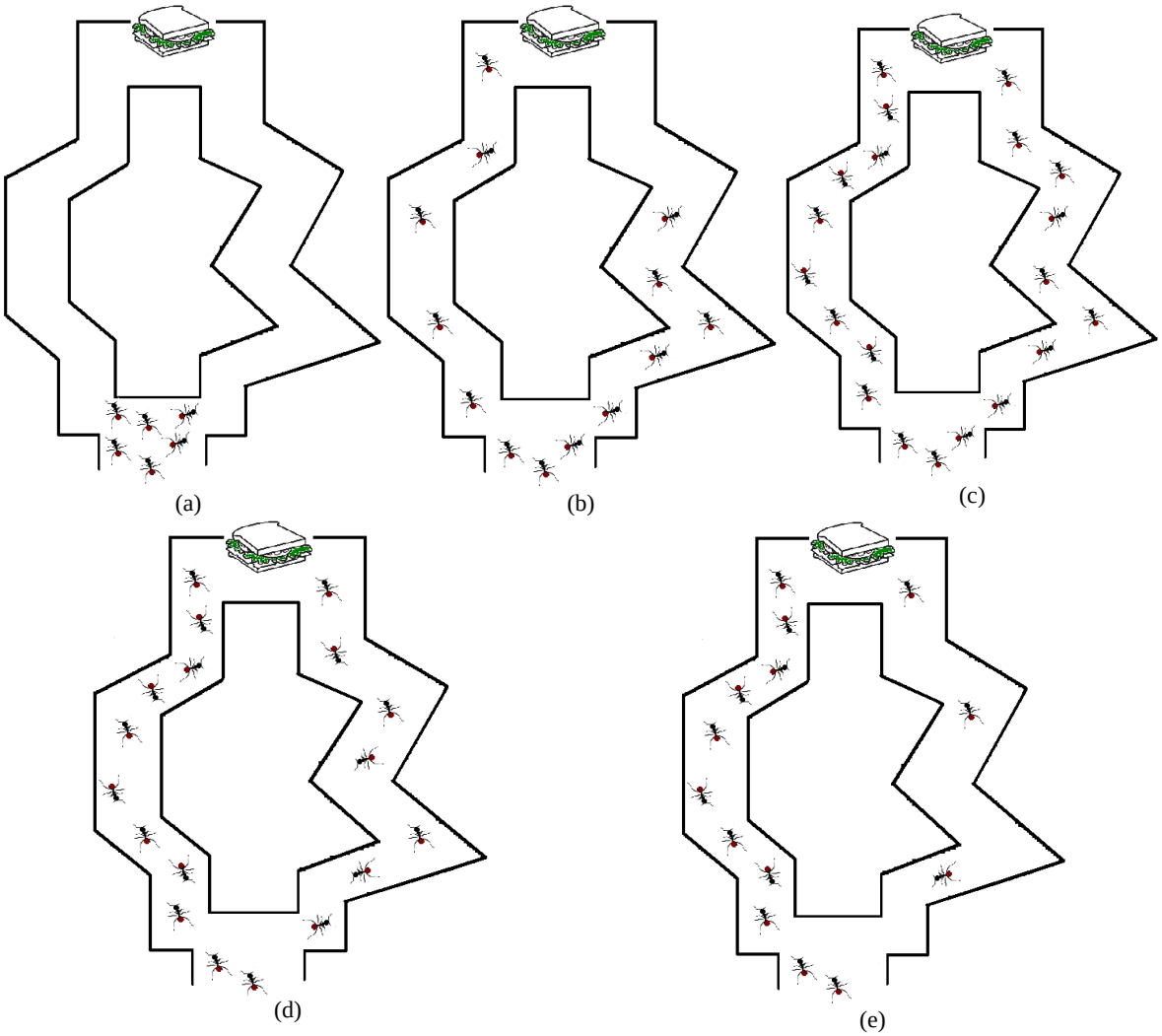


Figura 2.3. Elección del camino más corto.

En la Figura 2.4 se puede observar este comportamiento. En este caso el camino elegido durante la ejecución del algoritmo mostrado en la Figura 2.3 se obstruye con la aparición de un obstáculo. Las hormigas tratan entonces de seguir el camino anterior (Figura 2.4 (a)) mientras

algunas se escapan y van por el camino más largo guiadas por la probabilidad de elección. Al ocurrir esto, llegará un momento en el que las hormigas depositarán tal cantidad de feromona que el camino largo tendrá más probabilidad de elección que el camino más corto actualmente obstruido (Figura 2.4 (b)), pues la feromona disminuirá en el camino corto (por el proceso de evaporación) y aumentará en el largo (por la actualización global).

Este proceso finalizará con el momento mostrado en la Figura 2.4 (c), en el cual prácticamente todas las hormigas van por la alternativa válida mientras alguna sigue tratando de ir por el otro lado, posibilitando que si en algún momento el camino se restableciera pueda volver a ser elegido gracias a no tener una probabilidad de elección nula.

Ambos comportamientos (elección del camino más corto y adaptación a cambios en el entorno) se observan en todas las versiones de los algoritmos ACO.

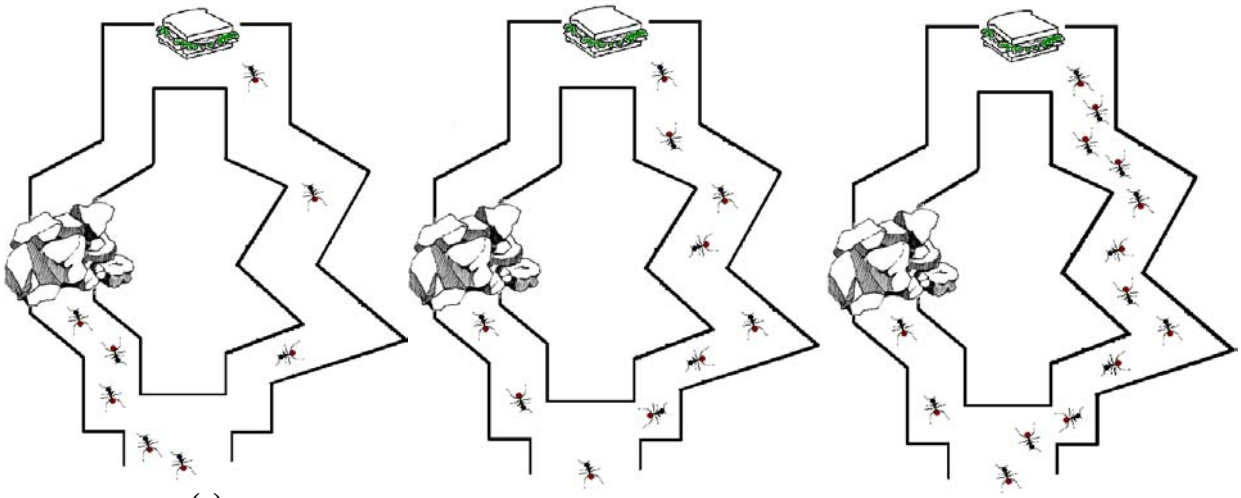


Figura 2.4. Recuperación ante cambios.

2.3.1.2 Sistema de Hormigas con Elitismo (Elitist Ant System)

Esta versión introduce una variación al algoritmo básico consistente en premiar a aquellas hormigas, un total de *e* individuos, que son mejores que el resto (que han obtenido mejores soluciones que el resto), es decir, refuerza su efecto en la búsqueda [Dorigo et al., 1996]. A estas hormigas se las denominará *hormigas elitistas*.

$$\tau_{ij} = e \cdot \frac{cte}{L} + \tau_{ij}$$

Ecuación 2.5. Incremento de feromona para las hormigas elitistas.

Con el fin de incluir este refuerzo en la búsqueda, aquellos enlaces que formen parte de los caminos encontrados por las hormigas elitistas verán su feromona incrementada siguiendo la Ecuación 2.5, donde *L* es la longitud del mejor camino encontrado.

2.3.1.3 Sistema de Colonia de Hormigas (Ant Colony System)

Publicado en [Dorigo and Gambardella, 1997], y basado en la extensión del algoritmo base teniendo en cuenta la utilización de técnicas de aprendizaje [Gambardella and Dorigo, 1995; Dorigo and Gambardella, 1996], el algoritmo *Ant Colony System* introduce las siguientes modificaciones con respecto a las versiones anteriores de los algoritmos basados en colonias de hormigas:

- Incluye una regla para elegir el siguiente nodo al que moverse que balancea la exploración de nuevos nodos del grafo y la explotación del conocimiento acumulado, lo que permite estudiar nuevas soluciones intentando mejorar las ya existentes. Para hacer esto, se utiliza un nuevo parámetro llamado q_0, un valor aleatorio en el intervalo *[0, 1]* llamado q, y se aplica la Ecuación 2.6 para saber la probabilidad de elegir un determinado nodo como siguiente punto del camino. En esta ecuación, el término *P* hace referencia a la Ecuación 2.1.

$$p_{ij}^{a} = \begin{cases} \arg\max_{x \in Adj(i)} \{[\tau_{ix}][\eta_{ix}]^{\beta}\}, si\ q \leq q_0 \\ P \qquad\qquad\qquad\qquad\qquad\quad , si\ q > q_0 \end{cases}$$

Ecuación 2.6. Probabilidad de elegir el siguiente nodo.

- La actualización global es realizada sobre el mejor camino encontrado (únicamente sobre los enlaces que lo componen), no sobre todos los enlaces de todos los caminos como ocurría en las versiones anteriores.

- Además de la actualización global de feromona, también habrá una local (*Actualización Local*), que ejecutará la evaporación y el depósito de feromona en cada enlace que se atraviese para alcanzar el nodo destino.

2.3.1.4 Sistema de Hormigas con Máx-Mín (Max-Min Ant System)

Esta variación introducida por [Stützle and Hoos, 1998, 2000] impone una cota máxima ($\tau_{máx}$) y una mínima ($\tau_{mín}$) a la cantidad de feromona que puede existir en cada enlace del grafo con el fin de evitar la convergencia prematura a una solución, es decir, $\forall\tau_{ij}$, su valor estará en el intervalo $[\tau_{mín}, \tau_{máx}]$. Con esto, se impone que la probabilidad de elección de un nodo esté siempre por encima de un umbral determinado.

2.3.2 Aplicaciones

Una vez se han estudiado las características principales de las versiones más relevantes, o conocidas, de los algoritmos basados en colonias de hormigas, se pasan a citar ejemplos de aplicaciones en distintos dominios, ya que aunque este tipo de algoritmos fue inicialmente pensado para ser aplicado en el problema del viajante, se ha empleado en gran variedad de escenarios para resolver problemas muy diversos [Dorigo et al., 1999; Mullen et al., 2009],

centrándose todos los trabajos en calcular el camino entre dos nodos del grafo que formaliza el problema:

- Problemas matemáticos: el problema de la asignación cuadrática [Maniezzo and Colorni, 1999], el problema del ordenamiento secuencial [Gambardella and M. Dorigo, 2000], y el enrutamiento en redes de telecomunicaciones [Schoonderwoerd et al., 1996; Dorigo and Di Caro, 1998].
- Problemas de planificación: el problema del ascensor [Molina et al., 2007], la resolución del problema del *Minimum Tardy Task* [Alba et al., 2007], y la planificación de tareas balanceadas [Chang et al., 2009].
- Enrutamiento o cálculo de trayectorias: enrutamiento de vehículos (*Vehicle Routing Problem*) [Bullnheimer et al., 1999; Gajpal and Abad, 2009; Lee et al., 2010; Yu et al., 2009], cálculo de trayectorias para vehículos no tripulados [Ma et al., 2007], cálculo de rutas dentro de un museo [Jaén et al., 2011], y cálculo de rutas en el campo de batalla para vehículos militares minimizando el coste del mismo y maximizando la seguridad [Mora et al., 2009].
- Problemas de ingeniería del software: empleo de algoritmos basados en colonias de hormigas para mejorar los modelos de predicción de calidad del software [Azar and Vybihal, 2011] o de gestión de proyectos [Abdallah et al., 2009].
- Diseño: de circuitos lógicos [Mendoza, 2001] o de antenas [Quevedo, 2010].
- Problemas de optimización multiobjetivo [Xiao-hua et al., 2005] y optimización en problemas con parámetros de valor continuo en lugar de discreto [Chen et al., 2003].
- Cálculo del camino óptimo [Fan et al., 2004; Kolavali and Bhatnagar, 2009].
- Creación de una red de telecomunicaciones, es decir, saber dónde colocar los diversos *switch*, así como unir unos con otros [Ibrahim, 2006; Ibrahim and Al Harbi, 2008; Ibrahim et al., 2005].
- Creación de un sistema de refuerzo para una red eléctrica [Favuzza et al., 2006]: ver qué caminos tomar para proveer energía a los usuarios empleando el menor coste posible teniendo en cuenta ciertas limitaciones técnicas.
- Reconocimiento de perfiles médicos de riesgo [Ramos et al., 2009].
- Utilización de algoritmos de hormigas para aprendizaje inteligente con el fin de realizar una extracción de características de usuarios [Wang and Li, 2009].

No obstante, a pesar de la cantidad de trabajos que se han mostrado, todos ellos limitan el número de nodos/enlaces del grafo sobre el cual se realiza la búsqueda de caminos a varias

decenas o centenas, dando muestras de la complejidad de aplicar los algoritmos basados en colonias de hormigas a grafos mayores. Por ejemplo, en [Chen and Ting, 2010] trabajan con grafos con menos de 500 nodos para resolver el problema del enrutamiento de vehículos y ya lo consideran de gran escala y tienen que dividir el grafo en subgrafos más pequeños para poder obtener una solución. El principal problema de la aplicación de este tipo de algoritmos en grafos vastos radica en que cuantos más pasos tenga que dar una hormiga para alcanzar el nodo destino, mayor probabilidad existe de que se pierda (de que se aleje de la meta), y por tanto de que su proceso sea infructuoso o de que si alcanza una solución la calidad de la misma sea baja, pues son soluciones encontradas de manera aleatoria.

Puesto que los grafos están adquiriendo grandes tamaños (tal y como se ha explicado en apartados anteriores), podría ser interesante encontrar una forma de adaptar los algoritmos basados en colonias de hormigas para trabajar sobre ellos pudiendo aprovechar las distintas características de interés que poseen como su rápida adaptación a cambios o como, por ejemplo, el hecho de que este tipo de algoritmos no requiere de conocimiento previo del grafo como ocurre en otros muchos algoritmos metaheurísticos cuya calidad del resultado depende de la buena elección de la función heurística que se utilice, que no siempre existe y que requiere, en muchos casos, el empleo de gran cantidad de tiempo para encontrar una adecuada al problema/dominio a tratar.

Algunos autores ya se han dado cuenta de lo interesante de este tema y han desarrollado diversas variaciones de los algoritmos base para hacer viable su aplicación en grafos vastos estáticos y en grafos dinámicos.

Con respecto a la aplicación de los algoritmos de colonias de hormigas a grafos vastos, cabe destacar el trabajo desarrollado por Francisco Chicano y Enrique Alba que, después de indicar en [Chicano and Alba, 2008b] la necesidad de crear algoritmos para trabajar con grafos vastos (en concreto en *Model Cheking* para aplicaciones concurrentes), proponen una variación de ACO y su integración con herramientas de *Model Cheking* [Alba and Chicano, 2007a, 2007b; Chicano and Alba, 2008a, 2009].

Lo que hacen es variar ACO para poder trabajar con problemas de *Model Cheking* concurrente en los cuales no se sabe el número de nodos (pues al ser concurrente los nodos varían en función del camino tomado) ni dónde está el nodo destino. Es decir, no se pueden aplicar determinadas heurísticas. Además, los grafos que modelan este tipo de problemática tienen un gran tamaño (millones de nodos), por lo que no caben enteros en memoria principal. Con el fin de tratar con todos estos problemas, lo que hacen es ejecutar ACO y permitirle buscar mientras que no superen una longitud de camino tope. De manera que si se alcanza dicho tope de longitud y no se ha encontrado el objetivo tienen en cuenta dos alternativas: aumentar el tope

(λ), y si no es suficiente, volverlo a ampliar cuando así lo requiera la ejecución del algoritmo; borrar todo el camino que llevan las hormigas y empezar la búsqueda desde los nodos donde se quedaran, es decir, crear coronas concéntricas de búsqueda en torno al nodo inicio.

Junto a este proceso, con el fin de ahorrar espacio, comprimen los nodos o los quitan de memoria y los apuntan con punteros. Además, eliminan la feromona sin influencia alta.

Con todo esto consiguen resultados competitivos, es decir, dejan claro que es posible aplicar este tipo de algoritmos a grafos vastos cuando se incorporan una serie de modificaciones, pero plantean el problema de que si no reflejan toda la información del proceso realizado hasta la fecha, por ejemplo la información acumulada de la feromona, si existe un cambio en el grafo no se podrían adaptar al mismo de una manera rápida. Es decir, pierden la capacidad de adaptación propia de los algoritmos ACO.

Sin embargo, esta característica es muy importante, ya que hay gran cantidad de redes que cambian determinadas características con el tiempo (poseen dinamismo en mayor o menor grado), incluidas las redes sociales y otras redes de gran interés en la actualidad. Por este motivo, es primordial mantener las propiedades de los algoritmos ACO que les permiten recuperarse ante cambios en el entorno.

Con el fin de demostrar el buen comportamiento de los algoritmos basados en colonias de hormigas en entornos dinámicos, que se ha podido observar con el ejemplo de la Figura 2.4, y comprobar si existen propuestas de ACO para grafos vastos dinámicos, se va a proceder a realizar un estudio de los trabajos existentes en este dominio que se han considerado más relevantes. Para ello, atendiendo a la clasificación que se había realizado al principio del apartado, existen diversos autores que han tratado de ampliar los algoritmos de ACO presentados para tener en cuenta el hecho de que pueden existir variaciones estructurales en los grafos en los que buscan la solución o cambios en los costes de los nodos/enlaces. A continuación se muestran algunos de ellos:

- Problema del viajante con dinamismo (*Dynamic Travel Salesman Probem*): en este tipo de problemas, las ciudades aparecen y desaparecen [Guntsch and Middendorf, 2001; Guntsch et al., 2001], o el tiempo que se tarda en ir de una ciudad a otra cambia, es decir, el coste de los enlaces que unen las ciudades sufre variaciones [Eyckelhof and Snoek, 2002]. En este último también se simula la desaparición de ciudades. En ambos casos, la adaptación consiste en el manejo que se haga de la feromona que se haya depositado hasta el momento durante todo el proceso de búsqueda. En el caso de la variación en los costes, lo que se hace es establecer un límite inferior a la feromona para que no baje de un cierto umbral y todos los caminos tengan una cierta probabilidad de ser elegidos (para tener caminos alternativos disponibles), y suavizar la subida de la misma para que no

haya una excesiva diferencia entre todos los enlaces. Respecto al caso de la aparición/desaparición de ciudades, cuando una nueva ciudad aparece en el grafo, su feromona tiene que ser iniciada y afectar de alguna manera a la que tenían asociada las antiguas ciudades, afectando en mayor manera a aquellas que están cercanas a la nueva ciudad y en menor medida a las más alejadas. En cuanto al tiempo que requiere esta adaptación, tal y como indican [Angus and Hendtlass, 2005], toma poco tiempo, siendo éste menor que el requerido si se tuviese que reiniciar la búsqueda.

- Enrutamiento de vehículos en entornos dinámicos (*Dynamic Vehicle Routing Problem*): en este caso, aparecen nuevos usuarios que deben ser incluidos en la ruta a calcular una vez que el proceso de búsqueda de caminos ha comenzado. Ejemplos de adaptación a este tipo de problema se encuentran en [De Oliveira, 2009; Montemanni et al., 2005]. Al igual que ocurría en el caso anterior, la adaptación se resuelve mediante una correcta modificación de la feromona que se ha ido depositando durante el proceso de búsqueda. En concreto, en este tipo de problemas se plantean dos alternativas: inicializar la feromona, o reducir su cantidad sólo en aquellos enlaces próximos al cambio, siendo esta última opción la más habitual y la que mejores resultados ofrece.
- Planificación de tareas en aplicaciones con cierto paralelismo. Cuando se realiza una planificación de tareas en aplicaciones con paralelismo, puede ocurrir que los tiempos de planificación varíen mientras que se está buscando el camino óptimo para llevar a cabo todas las tareas. Con el fin de tratar con este tipo de dificultades, [Ren and Yun, 2008] proponen la incorporación de una fase de adaptación que tiene en cuenta estos cambios en la búsqueda.
- Cálculo del camino óptimo [Sousa and Bento, 2009] en redes de carreteras. En este caso el coste de recorrer cada enlace varía por la aparición de caravanas (o tráfico lento). Para tratar con esto, la feromona depositada se va actualizando teniendo en cuenta la información que van recogiendo las hormigas (que representan a los usuarios).

No obstante, las aplicaciones de ACO en entornos dinámicos no solo se restringen a las ya explicadas en la parte estática en las cuales se incorpora algún tipo de modificación temporal, sino que también pueden aplicarse en entornos estrictamente dinámicos. Un ejemplo de esto es la aplicación de los algoritmos ACO sobre redes ad hoc. En dichas redes los cambios se producen cada pocos segundos, de modo que exigen utilizar algoritmos con una capacidad de adaptación a cambios muy rápida. Ejemplos de esto son los trabajos siguientes:

- [Di Caro et al., 2005]: el grafo más grande con el que trabajan es de 1.500 nodos. Crean un algoritmo ACO que busca caminos en la red *MANET-mobile*. Para ello, dividen el algoritmo en distintas fases. En la primera de ellas, lanzan unas hormigas reactivas que

buscan caminos al destino y guardan en cada nodo los nodos adyacentes que poseen. En la segunda, el envío de paquetes se basa en la información de los nodos adyacentes almacenada en cada nodo, de manera que en caso de existir varias alternativas se elige una aleatoriamente. Además de esto, se utilizan hormigas proactivas para buscar caminos mejores que los encontrados y mantener en un estado consistente la información almacenada en los nodos. En cuanto al dinamismo, cuando se cae un nodo, se borra de la lista de sus nodos adyacentes y se lanzan las hormigas reactivas.

- [Cauvery and Viswanatha, 2008]: trabajan sobre grafos de cinco nodos y proponen aplicar ACO para buscar los caminos y actualizar la feromona tanto en la ida como en el retorno. En cuanto a la recuperación ante caídas de enlaces, se apoya en el hecho de que ACO proporciona múltiples caminos alternativos, de modo que se puede coger otro cuando el mejor camino guardado se vea afectado.
- En el caso de [Wang et al., 2009], el grafo tiene 200 nodos como máximo. Para tratar con dicho grafo, lo que hacen es dividirlo en zonas y calcular los caminos practicables dentro de cada zona y entre zonas. De este modo, cuando hay un cambio en el grafo, si afecta solo a una zona, se realiza una búsqueda de caminos dentro de esa zona de manera proactiva; y si es entre zonas, se buscan los caminos y se cambia la información de las intrazonas de manera reactiva si fuese necesario. Respecto a la búsqueda de caminos, se emplea ACO, y se deposita feromona en cada paso hacia el destino y en el retorno al nodo inicio.
- [Kadono et al., 2010]: utilizan grafos con un tamaño máximo de 150 nodos. La idea es aplicar ACO para saber por dónde enviar los paquetes empleando la feromona depositada por las hormigas. La distribución de la misma se basará en la robustez del enlace, de manera que se crearán caminos que tengan validez durante el mayor tiempo posible. Para ello, la feromona depositada depende de la probabilidad de que el nodo adyacente desconecte. Para calcular dicha probabilidad se tendrán en cuenta tres parámetros almacenados asociados a cada nodo adyacente: posición global, velocidad, y tiempo global de actualización. Además, cuando un nodo se desconecta, la feromona se distribuye entre sus nodos adyacentes teniendo en cuenta también estos tres factores, de manera que el camino sea fácilmente recuperable (hay más posibilidad de que no caiga el camino si se elige el más robusto).

La gran cantidad de trabajos existentes sobre grafos dinámicos y los buenos resultados obtenidos, dejan patente lo apropiado de la aplicación de este tipo de algoritmos a grafos dinámicos, aunque en todos los casos el tamaño del grafo sobre el que trabajan no supera el millar.

En suma, tras el análisis realizado de las distintas aplicaciones en las que se han empleado los algoritmos basados en colonias de hormigas, se puede concluir que presentan un buen comportamiento en grafos vastos estáticos y en grafos dinámicos de pequeño tamaño.

Con respecto a la primera conclusión, las propuestas basadas en ACO incorporan algún tipo de procesamiento para que no se tengan que recorrer grandes distancias de manera aleatoria obteniendo caminos de baja calidad. En cuanto al segundo tipo de grafo, los algoritmos ACO se apoyan en su imitación del comportamiento de las colonias de hormigas en la naturaleza, ya que éstas son capaces de adaptarse a cambios en el entorno y recuperarse ante ellos volviendo a encontrar el camino óptimo entre el hormiguero y la comida.

No obstante, no existen hasta la actualidad trabajos que apliquen este tipo de algoritmos para encontrar caminos en grafos vastos dinámicos. Es decir, trabajos que unan los buenos resultados mostrados por los algoritmos ACO en cada uno de estos dos entornos.

Esto, por ejemplo, sería sumamente útil en redes dinámicas con topología *Small-World* (como las redes sociales), ya que el número de pasos que es necesario dar para ir del nodo inicio al nodo destino es muy bajo, y por tanto las hormigas se encontrarían en un entorno favorecedor para su funcionamiento.

2.4 Conclusiones

A lo largo de este capítulo se han mostrado distintos algoritmos que pueden ser utilizados a la hora de buscar caminos dentro de grafos. Este recorrido va desde los algoritmos clásicos (como Dijsktra o el algoritmo de búsqueda en profundidad) hasta los algoritmos metaheurísticos, y especialmente los algoritmos basados en colonias de hormigas por ser aquellos que mejor se adaptan a algunas de las características del problema enfocado.

En toda esta exposición se ha hecho especial hincapié en aquellos trabajos que permitían buscar caminos en grafos vastos, como las redes con topología *Small-World* propias de redes sociales y redes *peer-to-peer*, y aquellos que se adaptaban a cambios estructurales o cambios en los costes de los nodos/enlaces.

A pesar de que existen multitud de propuestas en uno y otro campo, no se han encontrado soluciones que cubran todos los requisitos: búsqueda rápida de caminos en grafos vastos dinámicos. Algunos lo plantean, pero se centran más en una cuestión de mantenimiento de las estructuras creadas durante el preprocesado que en los cambios que puedan existir durante la búsqueda del camino en sí misma.

Por ejemplo, aunque los algoritmos ACO poseen la característica inherente de la adaptación y de que son buenos cuando el número de pasos a dar entre el nodo inicio y el nodo destino es bajo, no se han encontrado propuestas de su aplicación en grafos dinámicos con topología

S*mall-World*. Un intento se observa en [Ahmad and Srivastava, 2008] pero los grafos de tipo *Small-World* empleados son estáticos y con menos de 100 nodos.

Todo esto se puede apreciar de forma gráfica en la Tabla 2.1, en la que se observa cómo ningún trabajo llega a cubrir todos los requisitos.

		Grafo Vasto	Grafo Dinámico	Topología Genérica	Adaptado a Almacenamiento Secundario	Pre-procesado	Tiempo de Respuesta Reducido	Coste Camino
Exhaustivo	Dijkstra (Chan and Lim (2007)			√				**Óptimo**
	Madduri et al., 2009 Chan and Zhang, 2001 Barrett el al., 2006	√		√		**Global**	√	**Óptimo**
	Chan and Lim, 2007	√		√	√	**Global**	√	**Óptimo**
	Chan and Yang, 2009		**Restringido**	√		**Global**		**Óptimo**
	Nannicini et al., 2010	√	**Conocido**			**Global**	√	**Óptimo**
	Ding et al., 2008 George et al., 2007		**Conocido**	√	√			**Óptimo**
	Bramandia et al., 2009		√	√	√	**Global**		**Óptimo**
Heurístico	Delling et al., 2009 Goldberg et al., 2007	√				**Global**	√	**Óptimo**
	Schultes and Sanders, 2007	√	**Restringido**			**Global**		**Óptimo**
	Kim et al., 2007		√				√	**Reducido**
Metaheurístico	ACO (Dorigo and Blum, 2005)			√			√	**Reducido**
	Di Caro et al., 2005		√	√			√	**Reducido**
	Chicano, 2007	√		√			√	**Reducido**

Tabla 2.1. Resumen del Estado del Arte.

Centrándose en que la propuesta de algoritmo obtenga caminos empleando un tiempo de respuesta reducido, se puede ver que esto solo es posible en los casos en los que el grafo no es vasto, o en los que siéndolo, se aplica un preprocesado global, lo cual imposibilita el emplearlos en grafos dinámicos. La excepción a esto último se encuentra cuando el dinamismo a abordar es conocido, es decir, se conoce de antemano los cambios que se van a producir y por lo tanto se pueden tener preprocesadas alternativas de camino o saber los costes que va a poder tomar un enlace a lo largo del tiempo. El problema es que existen escenarios en los que los cambios son aleatorios y no es posible predecirlos.

El único caso en el que el tiempo de respuesta es reducido sin limitar el dinamismo se da dentro de la aplicación de algoritmos metaheurísticos, y más concretamente de una versión de ACO, no obstante, el tamaño de estos grafos es pequeño.

En lo relativo al tipo de topología, existen varias propuestas que podrían funcionar sobre grafos genéricos, pues no tienen en cuenta la topología de estos para ejecutar los algoritmos. No obstante, no satisfacen de manera conjunta el resto de requisitos.

Por esta demostrada carencia de algoritmos que aúnen todos los objetivos descritos en este trabajo, una innovación interesante incluiría una propuesta que intente adaptar los algoritmos ACO (por ser los únicos que consiguen adaptarse a cambios empleando un tiempo de respuesta bajo) a búsquedas de caminos en grafos con todas las características citadas (grafos vastos dinámicos y con topología genérica) teniendo como objetivo prioritario el reducir el tiempo de respuesta para que pueda ser aplicable a cualquier dominio.

Capítulo 3 Fundamentación y Conceptos Básicos de la Propuesta

Los objetivos que se pretenden alcanzar en este trabajo de tesis doctoral, y que ya se establecieron en la introducción, son los siguientes: conseguir un algoritmo cuyo tiempo de respuesta sea mínimo, que sea aplicable a grafos vastos, con una topología genérica, que posea una gran capacidad de adaptación a cualquier cambio que aparezca en el grafo, que mantenga una independencia entre la gestión de la información almacenada y el algoritmo de búsqueda, y que sea eficaz en el manejo de los datos con los que se trabaja. La consecución de estos objetivos se basa principalmente en el almacenamiento del grafo en un sistema de base de datos y en la aplicación del algoritmo metaheurístico ACO adaptado a la búsqueda de caminos en grafos con alta cardinalidad.

Respecto a la decisión tomada sobre qué soporte de almacenamiento aplicar, se basó en el hecho de que se necesitaban guardar todos los nodos y enlaces de los grafos que modelasen los escenarios que se iban a utilizar, así como la información asociada a cada uno de estos elementos (por ejemplo, en el caso de una red social dicha información podría ser todo el perfil de la persona con sus datos personales, intereses, imágenes, etc.). Esto suponía cantidades masivas de datos, ya que el tamaño del grafo tenía que ser lo suficientemente grande como para simular una red real, superando por este motivo los cientos de miles de nodos/enlaces con toda la información asociada a cada uno de estos elementos. Debido a esta cantidad de información a almacenar, se optó por el soporte secundario para evitar los complejos tratamientos empleados por los trabajos mostrados en el estado del arte para reducir la cantidad de información y, además, para poder tener siempre disponible toda la información sin prescindir de parte por limitaciones de espacio.

Una vez tomada esta primera decisión, se debía elegir entre los distintos sistemas de almacenamiento secundario. El parámetro que entró en juego en este caso fue el tamaño del grafo junto con su dinamismo, así como que se debía garantizar la independencia entre la gestión de la información y el algoritmo de búsqueda para poder dividir el problema a tratar en dos y que el algoritmo de búsqueda no tuviese ningún tipo de limitación en la forma de buscar

gracias a que el problema del almacenamiento ya estuviese resuelto por otro lado (lo contrario a lo visto en el estado del arte, en el cual la mayoría de los trabajos se centran más en el manejo de la información, en crear estructuras auxiliares, por no tener una separación entre la problemática del almacenamiento y la búsqueda del camino). Con respecto a lo primero, se necesitaba un almacenamiento que aportase mecanismos sencillos para mantener la consistencia de los datos y para poder simular, de una manera fácil, los cambios de cualquier tipo que pudieran aparecer en una red real. Esto, y la necesidad de tener mecanismos de manejo de datos masivos eficientes, decantaron la balanza hacia las bases de datos, entre cuyas características se encuentran satisfechos los requisitos mencionados. Además, los sistemas gestores de bases de datos incorporan mecanismos de refinamiento que permiten llevar a cabo replicaciones de la información para poder utilizar todos los procesadores que se tengan disponibles, distribuir la información en distintas máquinas de manera transparente de cara al acceso a la misma, gestión de la memoria intermedia para disminuir el tiempo de acceso a los datos, etc. De manera que este tipo de gestión de la información en soporte secundario evita los problemas de tiempo de acceso y permite al algoritmo buscar caminos sin tener que preocuparse de la cantidad de información del grafo.

Teniendo decidido el soporte de almacenamiento y la gestión del mismo, el siguiente paso era decidirse por un algoritmo que permitiese realizar búsquedas eficientes y eficaces sobre los grafos con las características previamente definidas. Después de llevar a cabo el análisis expuesto en el estado del arte, se optó por emplear como algoritmo uno basado en colonias de hormigas. El principal motivo viene determinado por las conclusiones extraídas del estudio de trabajos de investigación relacionados con esta tesis doctoral. Estos algoritmos son altamente adaptables a variaciones en el grafo mientras se está realizando la búsqueda y a que, si se hacen los cambios oportunos, pueden trabajar sobre grafos vastos obteniendo buenos resultados. Por ello, lo que se va a proponer es una adaptación de los algoritmos ACO que tiene en cuenta comportamientos observados en la naturaleza. Es decir, se va a unir a un algoritmo inspirado en la naturaleza (en el comportamiento de las colonias de hormigas) una innovación que permite aplicarlo en grafos vastos que también está basada en un comportamiento natural.

Esta adaptación de ACO está basada en añadir información en el grafo para ayudar a las hormigas a encontrar caminos de manera similar a cómo los animales son capaces de encontrar una fuente de alimento utilizando el olor que desprende dicha comida. Es decir, los animales no necesitan ver la comida para saber que existe, ya que ésta desprende un olor que es perceptible a una determinada distancia de mayor o menor longitud en función del animal o insecto que la esté olfateando. Esto permite a los animales dirigir sus búsquedas de comida de manera exitosa, ya que tienen pistas de hacía donde dirigirse en lugar de elegir de manera aleatoria una dirección de entre todas las existentes.

Este mismo principio es el que se va a utilizar en el algoritmo propuesto, de manera que en el grafo en el cual van a tener que buscar las hormigas el nodo destino, van a encontrarse con zonas de olor que tendrán como fuente de comida un nodo que puede ser el destino que se buscaba, u otro nodo que servirá para ayudarle a encontrar el resto del camino porque existan otras hormigas que hayan llegado hasta él desde el destino deseado.

Es decir, en el grafo se depositará comida en una serie de nodos elegidos bajo ciertos criterios (o distribuidos aleatoriamente por el grafo) que ayudarán a las hormigas a alcanzar sus destinos bien porque los propios nodos comida sean los destinos, o bien porque sirvan como puntos de encuentro entre hormigas que vengan de distintos nodos inicio (nodos opuestos en el camino, pues unas empezarán desde el inicio buscando el destino y otras seguirán el camino inverso), de manera que a partir de producirse el encuentro de hormigas de distintos nodos inicio se tendrá un rastro de feromona que se podrá seguir hasta el destino.

Esta incorporación permitirá a las hormigas no tener que buscar en la totalidad del grafo, si no que su área de búsqueda se podrá ver reducida, ya que no estará obligada a encontrar el nodo destino para construir el camino solicitado, bastará con encontrarse con una hormiga que venga de él en un nodo comida/punto de encuentro.

Debido a este don olfativo asignado a las hormigas, el algoritmo propuesto se va a denominar *SoSACO (Sense of Smell ACO)*. En concreto, la propuesta presentada en este trabajo se basará en el algoritmo básico de ACO (con una serie de modificaciones), pero podría basarse en cualquiera de las posteriores versiones (ACO con elitismo, el sistema de hormigas con Máx-Mín, etc.).

Figura 3.1. Búsqueda de caminos con un nodo comida/punto de encuentro.

Un ejemplo del modo de funcionamiento puede observarse en la Figura 3.1. En ella, existe un único nodo comida/punto de encuentro. El olor de esta comida se difunde por el grafo alcanzando a parte de la gente que lo forma. Cuando una búsqueda de camino es solicitada, unas hormigas se sitúan en el nodo destino y otras en el inicio, y ambas buscan el nodo contrario. Durante este proceso de búsqueda, ambas se encuentran con la comida y la emplean como punto

de encuentro para saber cuál es la dirección buena para alcanzar el nodo destino, de manera que ya tendrán el camino global uniendo los caminos parciales encontrados por cada una de ellas.

Es importante dejar claro desde el principio, que el olor a comida será distinto a la feromona que depositan las hormigas para comunicarse con sus compañeras. Las diferencias principales son las enumeradas a continuación:

- Mientras que la feromona se deposita en los enlaces del grafo, el olor se depositará en los nodos.
- La feromona la irán depositando las hormigas como medio de comunicación con el resto de hormigas de la colonia. Por lo tanto, será una característica propia del algoritmo ACO, mientras que el olor será propia de *SoSACO*.
- Mientras que la feromona se deposita en proceso, durante la búsqueda del camino, el olor se deposita en preproceso.
- Mientras que las hormigas emplean la feromona continuamente para moverse, la única relación que tendrán con el olor será que lo podrán utilizar como ayuda para alcanzar el destino y que a través de su paso por él ayudarán a dispersarlo.

Para que todo lo indicado sea viable, en la Figura 3.2 se muestra un esquema de las distintas etapas que forman parte del algoritmo propuesto. Dichas etapas se explicarán de una manera genérica en este apartado, después de dar una definición formal del escenario sobre el cual se realizará la búsqueda de los caminos, para dejar claros aspectos y terminología importante para la comprensión del resto del documento.

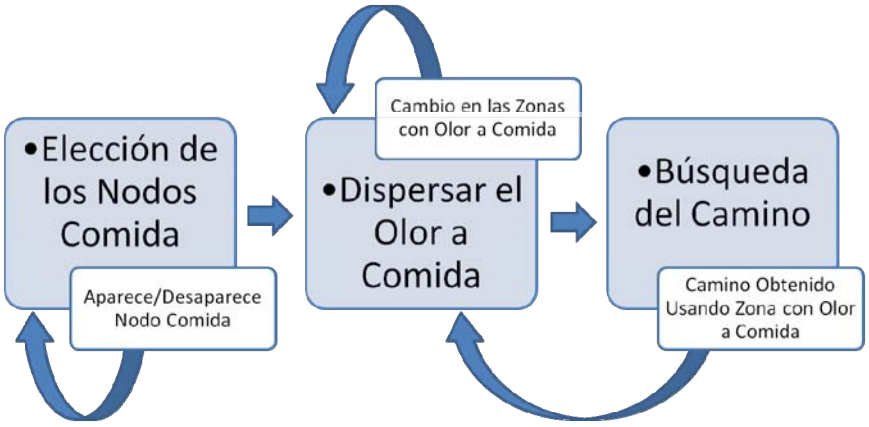


Figura 3.2. Etapas del algoritmo.

No obstante, antes de empezar con esta descripción, es importante explicar que aunque en la Figura 3.2 se muestra una secuencia de las distintas etapas una detrás de otra, esto no tiene por qué ser así: desde cada etapa se puede ir a cualquier otra, y durante la ejecución de cada una de ellas pueden llegar peticiones que impliquen ejecutar en paralelo otra etapa.

Todo este funcionamiento concurrente se puede apreciar en la Figura 3.3. En la misma, mientras se realiza la elección del nodo comida y su posterior dispersión del olor, se pueden resolver peticiones de búsqueda de caminos. Además, también puede apreciarse como ante cambios en el grafo, como es el caso del cambio en el coste de un enlace en el ejemplo de la Figura 3.3, la búsqueda del camino también prosigue sin verse interrumpida.

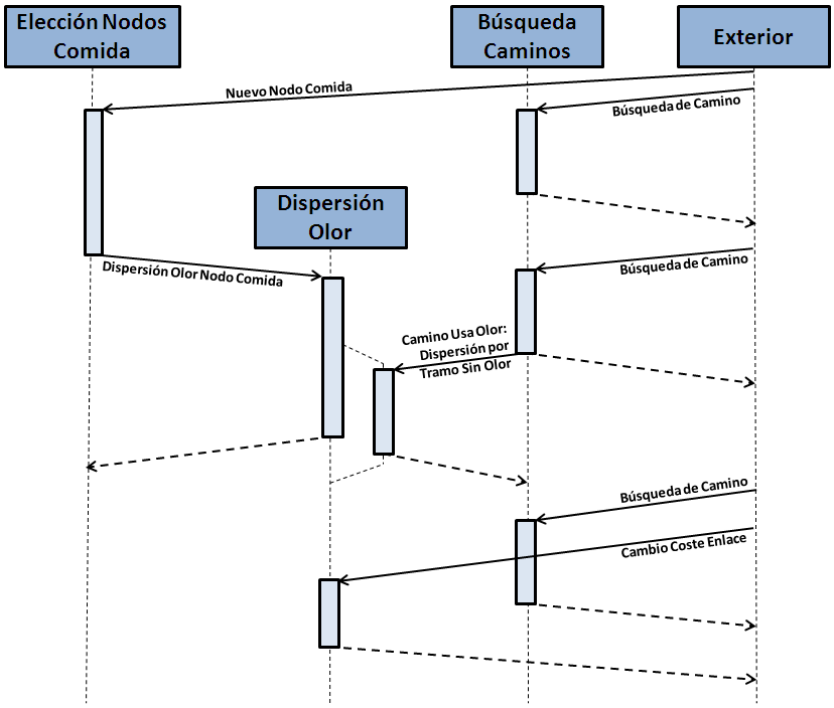


Figura 3.3. Ejemplo de funcionamiento.

Por último, resaltar que (como se ve en la Figura 3.2 y en la Figura 3.3), después de la búsqueda del camino puede, o no, ser necesario dispersar olor. Esto, como se explicará en el apartado 3.4, solo se hará cuando se utilice un nodo comida con su olor, al menos uno, para componer el camino global.

3.1 Formalización

Antes de pasar a describir cada una de las fases que componen el algoritmo propuesto, se va a formalizar el grafo sobre el cual se buscarán los caminos. De esta manera, dado un escenario, el mismo se puede representar por un grafo conexo $G(t) = \{N, L\}$ en un instante de tiempo t determinado, tal que $N(t) = \{i\}$ representa el conjunto de nodos que forman parte de G en el momento t, y $L(t) = \{l_{ij} = (i,j) \in N \times N, i \neq j\}$ es el conjunto de enlaces de G en el momento t, donde $l_{ij} = l_{ji}$.

Cada uno de los enlaces $l_{ij} \in L$ va a tener un coste que se puede definir con la función $W: L \rightarrow \boldsymbol{R}^+$, donde $w_{ij} = W(l_{ij})$.

En cuanto al tamaño del grafo, va a venir determinado por el número de nodos que posea, es decir, el número de elementos de N ($|N|$). En el caso de este trabajo, el tamaño del grafo será mayor de cientos de miles ($|N| > 10^5$).

Otra característica que posee G es su dinamicidad, es decir, para dos instantes diferentes de tiempo t_i y t_j donde $\Delta t = t_j - t_i \geq 0$ y su valor es del orden de milisegundos, se va a tener que el estado de G en cada uno de ellos ($G(t_i)$ y $G(t_j)$) va a ser distinto ($G(t_i) \neq G(t_j)$). La diferencia entre ambos estados va a poder venir dada por dos motivos diferentes: por un cambio en el peso de algún enlace tal que dado un $l_{ij} \in L$, $w_{ij}(t_i) \neq w_{ij}(t_j)$, o por un cambio estructural $|N(t_i)| \neq |N(t_j)|$ o $|L(t_i)| \neq |L(t_j)|$.

En un principio se podría pensar que para un Δt dado se daría el caso de una combinación de ambos tipos de modificación o de varios de cada uno de ellos. No obstante, al ser el tiempo una variable continua, por muchos que sean los cambios, se podrá decir que en media se hacen de uno en uno (uno por instante de tiempo), pasando a ser Δt un valor constante, y por tanto pudiéndose denominar periodo (T), cuyo valor será minúsculo en el caso de dinamicidad elevada o grande en el caso de que el grafo experimente pocos cambios.

Sobre G se van a solicitar una serie de servicios que requerirán la búsqueda de un camino entre cualesquiera dos nodos $i, j \in N$, tal que el camino obtenido $p_{ij} \in P$ ($P: N \times N \rightarrow L' \subseteq L$) deberá cumplir que su calidad se encuentre dentro de unos márgenes determinados, y que el tiempo que se tarda en obtenerlo (t_{answer}) sea menor que un tiempo fijado por el analista ($t_{threshold}$).

Una vez definido de una manera formal el grafo sobre el que se van a realizar las búsquedas de los caminos, se va a proceder a explicar, de manera genérica, cada una de las etapas que se muestran en la Figura 3.2, y será en el siguiente capítulo en el que se explicarán los detalles de la evolución del algoritmo en cada uno de los ciclos de la metodología que ha sido aplicada.

3.2 Elección de los Nodos Comida

Como se ha explicado anteriormente, los animales consiguen encontrar la comida de una manera eficiente utilizando su sentido del olfato. Esto es así porque todo alimento desprende un olor que puede ser detectado por sus futuros depredadores a una cierta distancia, habiendo una mayor probabilidad de detectar el alimento según se esté más cerca de él, pues el olor es más fuerte, y menor si se está más lejos, pues el olor que se pueda percibir será menor y por lo tanto será más complicado detectar dónde está la fuente de alimento.

Los dominios sobre los cuales se quieren realizar las búsquedas de caminos vienen representados por grafos de cientos de miles o millones de nodos. Esto implica que cada vez que se busque un camino se va a tener que explorar gran cantidad de datos hasta dar con el nodo

destino. Una forma de evitar esto sería seleccionar una serie de nodos en los que depositar "comida", de manera que sean fácilmente localizables por el olor que desprendan.

Las razones en las que se puede basar la selección de un nodo para realizar en él dicho depósito pueden ser las siguientes:

- Una elección basada en el grafo sobre el que se trabaja. En este caso se elegirán aquellos nodos que posean una determinada relevancia dentro del dominio que se esté representando (lo que Sanders o Goldberg denominaban *Landmarks* en el estado del arte): una persona famosa dentro de una red social (ya que va a tener más amigos que cualquier otra), un dispositivo con acceso a internet dentro de una red ad hoc (puesto que tendrá más dispositivos conectados a él), etc. La característica que se podría emplear para determinar estos nodos sería, por ejemplo, aquellos con un alto grado de centralidad dentro del grafo [Borgatti, 2005; Freeman, 1979] que vendrían a representar a los nodos con un alto valor del coeficiente de agrupamiento. Otras alternativas serían elegir a aquellos con un mayor grado o los que permitiesen dividir el grafo en zonas.
- Una elección basada en la experiencia extraída de los caminos solicitados en el pasado. En este caso, los nodos elegidos serían aquellos frecuentemente solicitados como nodos inicio o destino de caminos, o que suelen ser utilizados como nodos intermedios de distintos caminos (nodos con alta frecuencia de tránsito). Esta razón puede incluir una adaptación automática, ya que los nodos elegidos pueden ir variando en función de las sucesivas peticiones de camino, de manera que si el tipo de solicitud varía, también lo harán los nodos en los cuales se deposita la comida. Es decir, en este caso se añadiría un aprendizaje automático.

La forma de representar a estos nodos, y poderlos distinguir del resto de nodos pertenecientes a N, va a ser mediante la creación de un nuevo conjunto denominado F donde cada elemento viene representado por f_i. De manera que los distintos componentes de este subconjunto de N ($F \subseteq N$) van a satisfacer una de las razones indicadas anteriormente (o ambas).

3.3 Dispersión del Olor

Una vez se tienen elegidos los nodos comida, es preciso dotarlos de las características propias de las fuentes de alimento de la naturaleza, es decir, permitirles que su olor no quede recluido a él mismo sino que exista una dispersión en un área cercana, siendo dicho olor menos intenso según se está más alejado de la fuente de comida.

Esto, tal y como se ha explicado anteriormente, va a permitir disminuir el área de búsqueda, ya que utilizando el sentido del olfato, una vez localizado un rastro oloroso, solo habrá que

seguir dicho rastro en sentido creciente para encontrar de manera rápida la fuente de alimento evitando realizar comprobaciones extra en nodos que no aportan ningún tipo de información. De manera que mientras se utiliza un rastro las hormigas no se moverán de manera aleatoria, sino que lo harán empleando su olfato y elegirán como siguiente nodo el adyacente de mayor olor.

Con el fin de definir de una manera más formal está dispersión de olor, se van a pasar a definir una serie de conceptos implicados en dicha difusión:

- El olor asociado a cualquier $i \in N$ se denota por $O(i)$.
- El olor máximo que va a poder tener un nodo va a estar limitado, y va a ser igual al valor m fijado por el analista encargado de configurar el algoritmo.
- Los únicos nodos que van a poder poseer el valor máximo de olor van a ser aquellos pertenecientes a F ($O(f_i)=m$, $\forall f_i \in F$). Esto va a ser así porque los distintos f_i son las fuentes que desprenden olor, y por lo tanto deben ser las que mayor olor posean de todo el grafo.
- Con el fin de representar los nodos a los cuales llega el olor desprendido por los f_i se va a crear un nuevo conjunto denominado S que va a estar compuesto de tantos subconjuntos de N como elementos haya en F ($|S|=|F|$), es decir, $S = \{s_1, s_2,.., s_h\}$ donde $h = |F|$, y cada $s_i \subseteq N$. Esto va a ser así porque $\forall f_i \in F$ va a existir un conjunto de nodos asociados a los que llega su olor, pudiendo ir desde subconjuntos con un solo elemento (el propio f_i porque no haya nodos cerca a los que dispersar el olor) a subconjuntos con tantos elementos como nodos tenga el grafo ($|s_i|=|N|$).
- Todos los nodos con olor van a encontrarse dentro de S, pudiéndose dar el caso de que un mismo nodo esté en varios s_i, o en todos, ya que a un mismo nodo le pueden llegar todos los olores existentes.
- Cuando se realiza la difusión del olor de un f_i, todos los nodos pertenecientes a su s_i van a tener un olor comprendido entre m (cantidad de olor en f_i) y u. La variable u va a tener un valor impuesto por el analista y fijará un límite inferior de olor en la difusión del mismo, por lo que tendrá una influencia directa en el tamaño de las distintas áreas de olor. Es decir, u simulará el hecho de que el olor a comida no es perceptible a una determinada distancia.
- Cuando exista algún cambio que afecte a un s_i, el mismo será reiniciado (se evaporará el olor de algunos de los nodos que lo componen, o eventualmente de todos), de modo que se iniciará la dispersión desde la zona del cambio al resto de nodos mientras el olor a dispersar sea mayor o igual que u. Esto va a implicar que sea necesario realizar una buena elección del valor de u, ya que si el mismo es elevado, pocos cambios afectarán a

cada s_i porque el tamaño de los mismos será pequeño, pero este pequeño tamaño también implicará que será más difícil encontrar estas áreas, y no servirán de gran ayuda para encontrar la comida. No obstante, si el *u* es pequeño, se ayudará en mayor medida a encontrar la comida pero le afectarán muchos cambios (pues habrá más nodos que formen parte del s_i), y se tendrá que reiniciar en mayor número de ocasiones las áreas de olor, tomando gran cantidad de tiempo porque es necesario alcanzar a más nodos. Por lo tanto, será necesario encontrar un compromiso entre ambos beneficios. Además, a esto hay que añadir el hecho de que la propia dispersión de olor cuando el umbral es alto será más rápida que en el caso contrario, pues el número de nodos a alcanzar será menor.

```
 1: s_i, backpack, usedNodes = empty;
 2: n = f_i;
 3: odor(n), prevOdor = m;
 4: neighbors = reachedNodes(n);
 5: backpack.add(n);
 6: while (prevOdor > u) do
 7:  for (h = 1...neighbors.size) do
 8:    nAux = neighbors(h);
 9:    odorAux = odor(nAux);
10:    cost = W(n, nAux);
11:    odor = prevOdor – cost*k;
12:    if ((odor > odorAux) and (odor ≥ u)) then
13:      odor(nAux) = odor;
14:      if (!backpack.contains(nAux)) then
15:        backpack.add(nAux);
16:      end if;
17:    end if;
18:  end for;
19:  backpack.extract(n);
20:  usedNodes.add(n);
21:  n = nodeBiggestOdorInBackpack();
22:  if (∃n) then
23:    prevOdor = odor(n);
24:    neighbors = reachedNodes(n) - usedNodes;
25:  else
26:    prevOdor = u;
27:  end if;
28: end while;
29: s_i.add(usedNodes);
```

Algoritmo 3.1. Creación de s_i.

En cuanto al algoritmo seguido para la realización de la difusión del olor, es decir, para la creación de cada s_i, es el que se muestra en pseudocódigo en el Algoritmo 3.1. De dicho algoritmo es importante explicar de una manera detallada los siguientes aspectos:

- El Algoritmo 3.1 deberá ejecutarse por cada f_i existente, ya que todos los nodos comida dispersan olor para ayudar en las búsquedas posteriores de caminos.
- Ya que la intensidad de olor a comida disminuye según se está más alejado del nodo f_i, es necesario definir la fórmula a través de la cual el olor se ve reducido en función de la distancia que lo separa de f_i. Para ello, como indica la Ecuación 3.1, el olor asociado al nodo del que se parte *O(j)* va a verse reducido en el siguiente nodo *i* por una cantidad en la que la distancia que los separa w_{ij} se verá ponderada por un valor $k \in \boldsymbol{R}$ ($0 \leq k \leq 1$) fijado por el analista. Este proceso se puede apreciar en el Algoritmo 3.1 en la *línea 11*.

$$O(i) = O(j) - k \cdot w_{ij}$$

Ecuación 3.1. Dispersión del olor.

Es importante indicar que a la hora de fijar un valor para *m* así como para *k*, es necesario hacer un estudio para garantizar que la difusión del olor pueda llegar a cualquier nodo del grafo.

- Puesto que en la naturaleza se llega a un determinado umbral de olor que es imperceptible por cualquier animal, se fijará un valor para *u* tal que no se permitirá asignar a un nodo un olor menor que dicho valor (*línea 12* del Algoritmo 3.1)

Una vez explicados los términos de relevancia dentro del algoritmo de dispersión, se va a proceder a explicar el algoritmo en sí, es decir, los distintos pasos que se siguen en Algoritmo 3.1.

Lo primero que se realizará será la inicialización del s_i como un elemento vacío, así como una variable en la que se almacenarán los nodos a los que se les ha asignado olor y que deben tenerse como nodos desde los que se dispersará el mismo, si es posible, a sus nodos adyacentes (todo esto se realiza en la *línea 1*). Dicha variable se denominará *backpack*. Por último, en esta línea se inicia una variable denominada *used Nodes*. La función de ésta es tener almacenados todos aquellos nodos que han sido utilizados para extender su olor a sus nodos adyacentes.

A continuación se asignará a la variable *n* el nodo comida (*línea 2*). Dicha variable se utilizará como auxiliar para tener guardado el nodo que se está teniendo en cuenta para dispersar el olor en cada momento, y es por este motivo por el que inicialmente toma el valor del nodo comida, porque es del nodo del que se parte para dispersar el olor (y por este motivo será el primer elemento de *backpack* como se indica en la *línea 5*). Debido a que *n* en este momento es f_i, se le asignará el olor máximo (*m*) en la *línea 3*. En esta última línea también se asignará el olor máximo a *prev Odor*, que es la variable en la que se guarda el olor del nodo desde el que se dispersa el olor a sus nodos adyacentes.

Después de inicializar las variables auxiliares que se utilizarán en la dispersión, se pasará a difundir el olor. Para ello, lo primero será seleccionar todos los nodos adyacentes a *n* aplicando el método *reached Nodes (n)*. El resultado, un subconjunto de *N*, va a almacenarse en la variable *neighbors*. Esto se puede ver en la *línea 4* del Algoritmo 3.1.

A continuación, mientras que el valor del olor que se pretende dispersar en cada momento (*prev Odor*) sea mayor que *u* se irán seleccionando los nodos a los que hacer llegar el olor y se les asignará la cantidad correspondiente (*líneas 7-27*). Los pasos a realizar para ello son los siguientes:

a) Para cada nodo adyacente a *n* (para cada nodo almacenado en la variable *neighbors*) se calcula el olor que le llegaría partiendo de *n* aplicando la Ecuación 3.1, es decir, el olor que le llegaría al vecino *h*-ésimo (*neighbors(h) = nAux*) sería el olor del nodo *n* menos el coste del enlace que los une ponderado con el factor *k* (esta cantidad se almacena en *odor*). Dicho valor de olor se le asignaría a *nAux* siempre y cuando el olor que tuviese asignado no fuese mayor que el obtenido, y que *odor* fuese mayor o igual que el olor umbral *u*. Además, si *nAux* no está incluido dentro del subgrupo *backpack* para ser estudiado posteriormente como nodo desde el cual dispersar el olor, se le añadirá a dicho grupo de nodos. Esto se realiza en las *líneas 7-18*.

b) El nodo *n* se extraerá del subconjunto *backpack* (*línea 19*), para evitar que se vuelva a tener en cuenta como nodo desde el cual dispersar el olor, y se le marcará como nodo visitado incluyéndolo en el subconjunto *used Nodes* (*línea 20*).

c) Se elige el siguiente nodo desde el que dispersar el olor. Para hacerlo se aplicará la función *node Biggest Odor In Backpack*, que selecciona el nodo con mayor olor dentro del subconjunto de nodos *backpack*, y se asignará el nodo elegido a la variable *n* (acción realizada en la *línea 21*).

d) En caso de existir un nodo desde el que seguir dispersando, se seleccionarán sus nodos adyacentes que no hayan sido utilizados para dispersar olor desde ellos (*reached Nodes(n)-used Nodes*) y se asignarán a la variable *neighbors* para volver a realizar todo el proceso explicado (se vuelven a realizar los pasos desde el punto *a*). Si no existen más nodos desde los que dispersar, se parará el proceso (se asigna a la variable *prev Odor* el valor de olor umbral *u*).

Cuando se finaliza la dispersión, el subconjunto de nodos que formarán parte de s_i serán los que aparezcan en *used Nodes* (*línea 29*).

3.3.1 Ejemplo de Dispersión del Olor

Se va a partir de un grafo inicial, Figura 3.4 (a), que tiene once nodos, $N = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, y diez enlaces, $L = \{l_{1,2}, l_{1,5}, l_{1,8}, l_{1,9}, l_{2,3}, l_{4,11}, l_{5,4}, l_{5,6}, l_{5,10}, l_{9,7}\}$, cuyos pesos son los siguientes: $W = \{w_{1,2}, w_{1,5}, w_{1,8}, w_{1,9}, w_{2,3}, w_{4,11}, w_{5,4}, w_{5,6}, w_{5,10}, w_{9,7}\} = \{2, 2, 2, 2, 4, 3, 4, 5, 1, 4\}$.

Una vez definido el grafo, se pasan a mostrar las fases del algoritmo relacionadas con la comida y la dispersión de su olor:

- *Elección de los nodos comida*. Observando el grafo mostrado en la Figura 3.4 (a), puede deducirse que, en el caso de que representase una red social, el nodo *1* sería una persona importante, por lo que sería dicho nodo el seleccionado como nodo comida. Es decir, ese nodo pasaría a formar parte del conjunto *F* como f_1. De manera que $F = \{f_1\}$. La

forma de representar esto se aprecia en la Figura 3.4 (b), en la cual se puede ver como dicho nodo queda resaltado con respecto al resto, pues contendrá una gran cantidad de olor por ser la fuente de comida. En concreto, el olor que contendrá será igual a *m*: $O(1) = O(f_1) = m = 10$.

- *Dispersión del olor*. Se pasa a expandir el olor del nodo comida. Para ello, se va a considerar $u = 6$ y $k = 1$. El proceso que se sigue se muestra a continuación paso a paso:

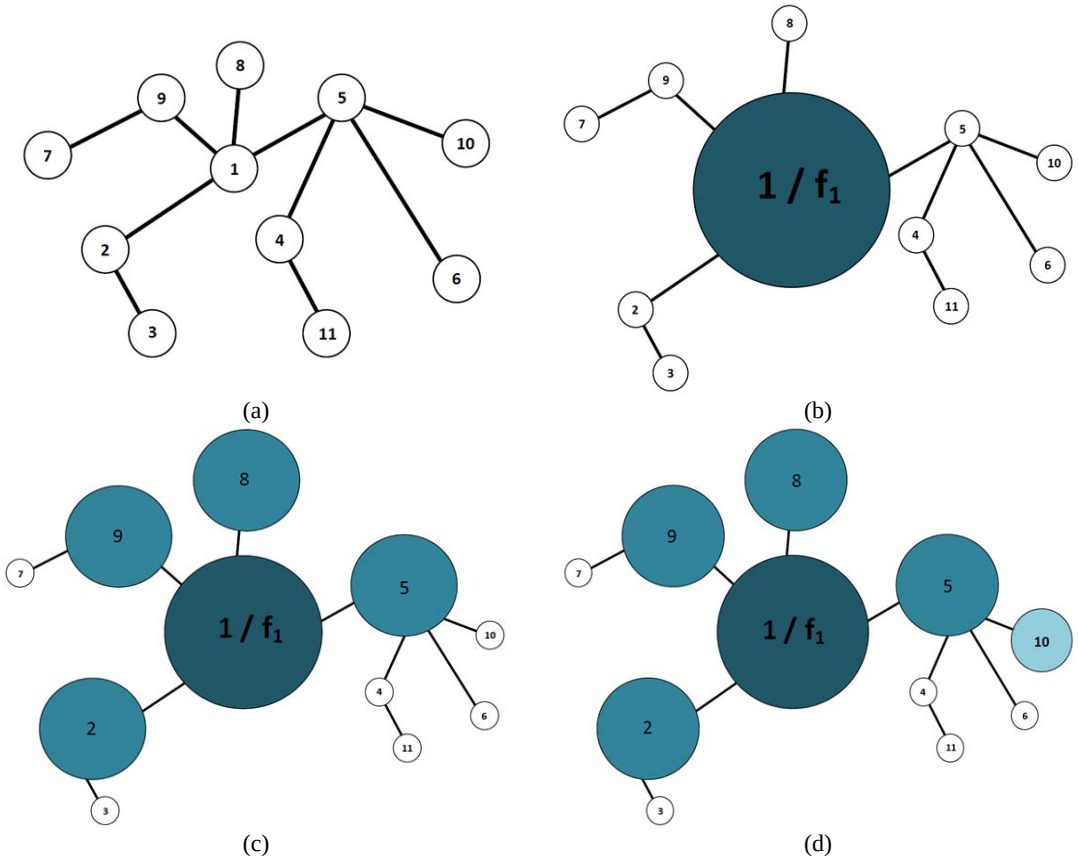


Figura 3.4. Dispersión del olor.

- Paso 1:
  - *backpack = {1}*
  - *usedNodes = {}*
- Paso 2:
  - Se selecciona el nodo de *backpack* con mayor olor, que en este caso es *1*, de modo que $n = 1$
  - $O(2) = O(1) - k \cdot w_{1,2} = 10 - 1 \cdot 2 = 8$
  - $O(5) = O(1) - k \cdot w_{1,5} = 10 - 1 \cdot 2 = 8$
  - $O(8) = O(1) - k \cdot w_{1,8} = 10 - 1 \cdot 2 = 8$
  - $O(9) = O(1) - k \cdot w_{1,9} = 10 - 1 \cdot 2 = 8$

- *backpack = {1, 2, 5, 8, 9}*
- *usedNodes = {1}*
- Los nodos a los cuales se les asigna olor en este paso (a los que les llega olor directamente desde el f_1) se les ha coloreado de un color azul oscuro, aunque menos que el nodo comida, en la Figura 3.4 (c) para diferenciarlos del resto. Además han aumentado su tamaño con respecto al resto, aunque menos que el nodo comida, pues su olor es mayor que el de los nuevos nodos incorporados en *backpack*. Además, se puede observar que tanto el color como el tamaño de estos nodos es igual entre ellos. Esto es debido a que todos tienen el mismo olor.

o Paso 3:

- Se selecciona el nodo de *backpack* que no haya sido utilizado con anterioridad, y cuyo olor sea el mayor posible. En este caso, como hay varios con el mismo olor, se selecciona uno al azar: *n = 8*
- No hay nodos adyacentes a *8* no visitados con anterioridad, de modo que desde este nodo no se puede dispersar olor a otros.
- *backpack = {1, 2, 5, 8, 9}*
- *usedNodes = {1, 8}*

o Paso 4:

- Se selecciona el nodo de *backpack* con mayor olor y que no haya sido utilizado con anterioridad. Al igual que en el caso anterior, como hay varios con el mismo olor, se selecciona uno al azar: *n =2*
- $O(3) = O(2) - k \cdot w_{2,3} = 8 - 1 \cdot 4 = 4 < u = 6 \rightarrow O(3) = 0$
- Este nodo no se añadirá a *backpack* porque su olor es menor que *u*, de manera que ni él ni ninguno de sus nodos adyacentes (si solo se llega a ellos a través de él) podrían ser incluidos en s_1.
- *backpack = {1, 2, 5, 8, 9}*
- *usedNodes = {1, 8, 2}*

o Paso 5:

- Se selecciona un nodo al azar entre todos aquellos de *backpack* que no han sido utilizados hasta el momento y que tienen el mismo olor máximo: *n=5*
- $O(4) = O(5) - k \cdot w_{5,4} = 8 - 1 \cdot 4 = 4 < u \rightarrow O(4) = 0$

- $O(6) = O(5) - k \cdot w_{5,6} = 8 - 1 \cdot 5 = 3 < u \rightarrow O(6) = 0$
- $O(10) = O(5) - k \cdot w_{5,10} = 8 - 1 \cdot 1 = 7$
- De todos los nodos adyacentes a *5* solo se añadirá a s_1 el nodo *10*, ya que el resto tendría un olor menor que *u*, hecho que no es admisible en la dispersión. Esta nueva inserción en s_1 se aprecia en la Figura 3.4 (d), en la que se ve el nodo *10* más grande que aquellos sin olor y de un color azul claro. Dicho color es más claro que el de los otros nodos de s_1, y su tamaño es menor. Esto se ha hecho así para representar que su olor es menos fuerte que el de los otros nodos del conjunto s_1.
- *backpack = {1, 2, 5, 8, 9, 10}*
- *usedNodes = {1, 8, 2, 5}*

o Paso 6:

- Se selecciona el nodo con mayor olor de *backpack* que no haya sido utilizado con anterioridad: *n=9*
- $O(7) = O(9) - k \cdot w_{9,7} = 8 - 1 \cdot 4 = 4 < u \rightarrow O(7) = 0$
- Puesto que el olor que le correspondería a *7* es menor que *u*, no se le incluirá en s_1.
- *backpack = {1, 2, 5, 8, 9, 10}*
- *usedNodes = {1, 8, 2, 5, 9}*

o Paso 7:

- Se selecciona el único nodo que queda sin utilizar: *n=10*
- No hay nodos adyacentes a *10* no visitados con anterioridad, de modo que no se incluirán nuevos nodos en *backpack*.
- *backpack = {1, 2, 5, 8, 9, 10}*
- *usedNodes = {1, 8, 2, 5, 9, 10}*

Puesto que no hay más nodos en *backpack*, terminaría la dispersión de olor, siendo el resultado el siguiente: un nodo comida $F = \{ f_1 = 1\}$, y $S = \{s_1\} = \{\{1, 2, 5, 8, 9, 10\}\}$.

3.4 Primera Aproximación a SoSACO

Una vez que se han elegido los nodos comida y que se han dispersado los olores que les corresponden, se va a poder utilizar el olor como ayuda en el algoritmo de búsqueda de caminos entre dos nodos cualesquiera indicados por el usuario.

La forma en la que se va a realizar la búsqueda será utilizando un algoritmo básico ACO [Dorigo, 1992; Dorigo and Blum, 2005; Dorigo and Stützle, 2004] sobre el que se aplicarán una serie de modificaciones que irán variando a lo largo de la evolución del algoritmo para satisfacer el acercamiento progresivo a los objetivos establecidos inicialmente, siguiendo para ello la metodología que se explicará en el último apartado de este capítulo. Es decir, se va a aplicar un algoritmo de hormigas en el cual dichas hormigas van a estar dotadas de sentido del olfato recibiendo por este motivo el nombre de *SoSACO* (*Sense of Smell ACO*).

La utilización del olor es distinta en cada ciclo de evolución del algoritmo, pero en todos los casos se cumple que:

- Los nodos comida se pueden utilizar como "puntos de encuentro" entre hormigas. De manera que hormigas que vienen de nodos distintos pueden completar su camino si otra hormiga llegó a esa comida desde el nodo destino de la primera. Un ejemplo de esto se puede apreciar en la Figura 3.5, en la que la hormiga *a* y la hormiga *b*, partiendo de nodos del camino opuestos llegan al mismo nodo comida, pudiendo crear una camino global uniendo los siguientes caminos parciales: destino - nodo con olor encontrado por *a*, nodo con olor encontrado por *a* - f_i, f_i - nodo con olor encontrado por *b*, nodo con olor encontrado por *b* - inicio. Es decir, han empleado el nodo comida f_i como un punto de encuentro.

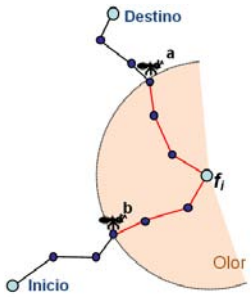


Figura 3.5. Nodo Comida empleado como punto de encuentro.

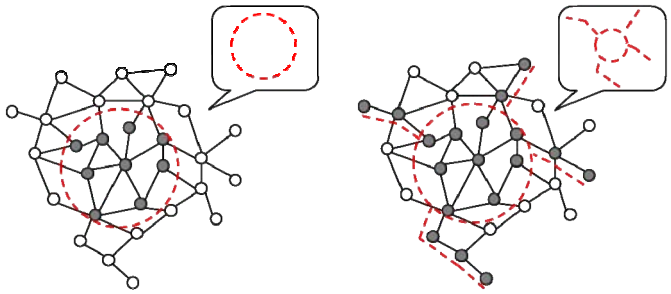
- Ir desde cualquier $i \in s_i$ a su nodo comida (f_i) es rápido, solo hay que seguir el rastro creciente de ese tipo de olor.

- Cuando una hormiga llega a un olor, el camino que ha seguido para ir desde su nodo inicio hasta ese nodo perteneciente a un s_i va a verse impregnado de olor si es el único camino hallado que va desde ese nodo inicio al tipo de olor encontrado (al olor correspondiente a un nodo comida determinado), o si no es el único pero el coste es menor que el de los otros.

  Dicha dispersión de olor se realizará cuando se termine la búsqueda del camino, es decir, cuando todas las hormigas hayan terminado de buscar. De este modo, los caminos que se plantean como candidatos a dispersar olor por ellos son los mejores encontrados para

cada nodo inicio/destino y nodo comida. Esto es así porque durante la ejecución del servicio se van almacenando los mejores caminos encontrados, de manera que si uno es mejor que otro ya almacenado con los mismos nodos extremos, se borrará aquel y se almacenará el nuevo.

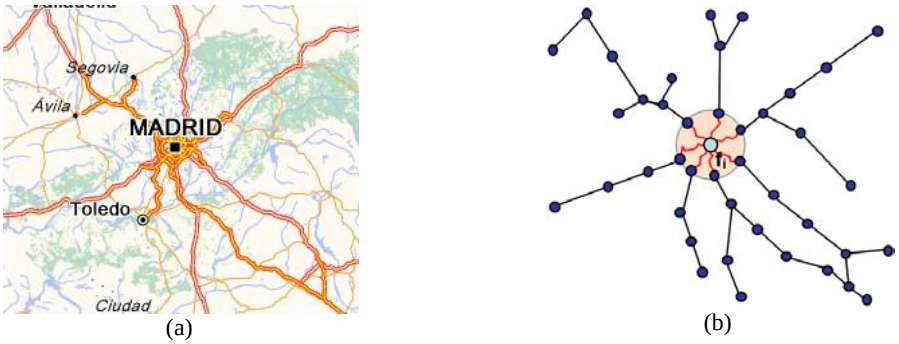
A este olor dispersado después de la dispersión inicial se le va a denominar *Olor Radial* ya que, como se puede apreciar en la Figura 3.6, se añade a la dispersión inicial Figura 3.6 (a), que en este caso se puede representar por un círculo, una serie de radios que representan los caminos encontrados por las hormigas que emplean el nodo comida Figura 3.6 (b).



(a) (b)

Figura 3.6. Ejemplo de dispersión radial frente a dispersión inicial.

Por ejemplo, suponiendo que el algoritmo se aplica sobre una red de carreteras como la mostrada en la Figura 3.7 (a), si se situase el nodo comida en la ciudad de Madrid, por ser el nodo que tiene una mayor cantidad de carreteras, el olor inicial se podría ver como el círculo de color rosáceo con centro en ella representado en la Figura 3.7 (b), de modo que cuando se empezase a buscar caminos, empezaría a aparecer olor radial asociado a las carreteras que atraviesan Madrid y que se representa mediante los nodos coloreados de azul en la Figura 3.7 (b).



(a) (b)

Figura 3.7. Ejemplo en una red de carreteras de la dispersión radial de olor.

- El olor solo es una ayuda para el algoritmo de búsqueda, de modo que su existencia/inexistencia no va a interrumpir en ningún momento la búsqueda de los caminos. El algoritmo de búsqueda no tiene que esperar a la finalización de un proceso de dispersión de olor para continuar con su búsqueda o empezar la misma, ya que el grafo no se ve modificado. Es decir, nunca va a suponer un incremento de los tiempos de

respuesta, su utilización disminuirá o igualará los tiempos obtenidos en el caso de no utilizarlo.

En cuanto al tercer punto, la forma de dispersar este olor radial se indica en Algoritmo 3.2. En él, la variable *path Start Odor* representa el camino que la hormiga ha encontrado desde su nodo inicio (primera posición en *path Start Odor*) hasta el nodo con olor (última posición en *path Start Odor*). De manera que el Algoritmo 3.2 recorrerá *path Start Odor* desde su última posición hasta la primera asignando a cada nodo (*path Start Odor (i-1)*) el olor que le corresponda (el olor del nodo anterior menos el coste del enlace que los une multiplicado por el factor *k*), es decir, el calculado en la *línea 3* del Algoritmo 3.2.

```
1:  for (i=pathStartOdor.size ... 2) do
2:    cost = W(pathStartOdor(i-1), pathStartOdor(i));
3:    odor = odor(pathStartOdor(i)) – k*cost;
4:    if (odor>0) then
5:      odor(pathStartOdor(i-1)) = odor;
6:    end if;
7:  end for;
```

Algoritmo 3.2. Dispersión de olor por el camino de la hormiga.

Como se puede apreciar en la *línea 4*, la asignación del olor solo se hará efectiva cuando el mismo sea mayor que cero, lo que supone una diferencia con respecto a la dispersión del olor en la creación de los s_i, pues en ese caso el umbral mínimo de olor que podía tener un nodo en la dispersión venía fijado por un valor $u \geq 0$. Esto va a permitir aumentar $|s_i|$ pudiendo llegar a alcanzar un valor igual al número total de nodos que forman el grafo. De manera que el nodo comida va a ser detectado por su olor desde cualquier nodo del grafo.

Aunque esta expansión podría suponer un problema en lo referente a la cantidad de cambios que afectan a las áreas con olor y que, por lo tanto, implican una re-inicialización de la dispersión del olor desde la zona afectada, trayendo consigo un aumento del tiempo requerido para dar una respuesta para el camino solicitado dado que durante la creación del área con olor la hormiga no dispondría de ningún tipo de ayuda para encontrar el destino. Esto no será así, ya que solo será necesario reiniciar el área con olor cuando el cambio afecte a nodos con $olor \geq u$, es decir, cuando el cambio afecte a aquellos nodos a los que había llegado el olor en la dispersión inicial.

En el caso de que el nodo afectado no formase parte de este subconjunto de nodos con olor inicial, no se reiniciará el área con olor afectada. El procedimiento que se seguirá será evaporar completamente el olor del nodo afectado por el cambio (x) así como el de todos los nodos con olor menor que x cuyo único camino para llegar al nodo comida tuviese que pasar obligatoriamente por x.

Por tanto, con este proceso se tendrá garantizado que el nodo comida será visible desde una gran distancia y, además, que en caso de existir cambios, la adaptación a los mismos requerirá una cantidad baja de tiempo.

3.4.1 Ejemplo de Dispersión de Olor Radial

En este ejemplo se pretende mostrar cómo, en algunos casos, se va a realizar una dispersión extra de olor, lo que se ha denominado olor radial, cuando se realizan búsquedas de caminos dentro de un grafo. Las búsquedas podrían solicitarse mientras se realiza la dispersión de olor, o la elección de los nodos comida, o después de éstas, ya que el algoritmo de búsqueda no se ve afectado por las fases anteriores. No obstante, en este ejemplo se va a partir del grafo en el estado mostrado en la Figura 3.4 (d), es decir, una vez finalizada la dispersión. El fin de esto es mostrar cómo influye el olor en la búsqueda de caminos.

Una vez ha quedado claro el estado en el que se encuentra el grafo cuando se realiza la petición del servicio, indicar que dicho servicio solicita la búsqueda del camino que une el nodo *11* con el nodo *1*. Es decir, se solicita $p_{1,11}$.

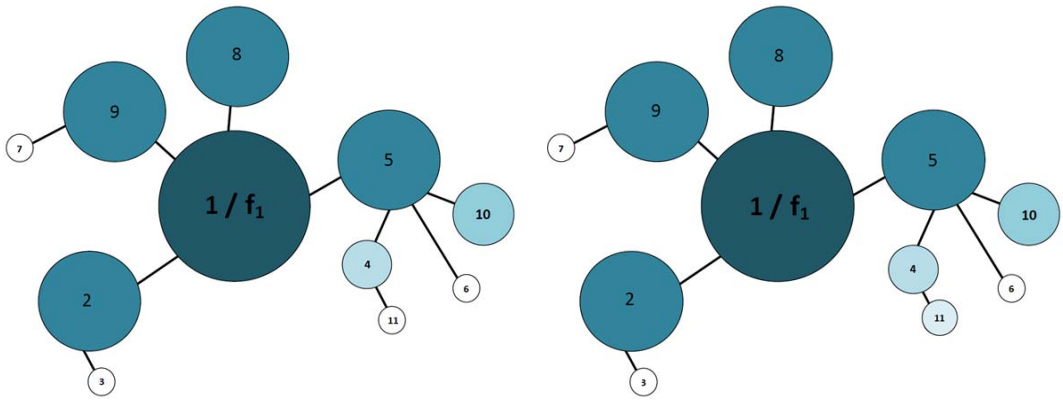
La búsqueda de ese camino termina cuando se encuentra un nodo con olor, ya que desde él es fácil llegar al nodo comida. En el caso del ejemplo, se encuentra el tramo {*11, 4, 5*}. Al llegar a *5*, como posee olor, la hormiga se para y busca como completar el camino siguiendo el rastro creciente de olor, de manera que completa el camino y obtiene $p_{11,1} = \{11, 4, 5, 1\}$.

Debido a que se utiliza el área con olor en el camino obtenido, una vez finalizado el servicio se procedería a expandir el olor por el trozo de camino que va desde el nodo inicio al primer nodo con olor. Para ello se siguen los siguientes pasos:

- Paso 1:
  - Primer nodo con olor encontrado: *5*
  - $s_1=\{1, 2, 5, 8, 9, 10\}$
- Paso 2:
  - Siguiente nodo: *4*
  - $O(4) = O(5)-k \cdot w_{5,4} = 8-1 \cdot 4 = 4$ (como es mayor que cero, este nodo se agregaría a s_1)
  - $s_1 = \{1, 2, 5, 8, 9, 10, 4\}$. Esta inserción se aprecia en la Figura 3.8 (a) con la aparición del nodo *4* coloreado y con el círculo que lo representa con un tamaño mayor que antes. Dicho color es azul, pero más claro que el resto de nodos con olor. Además, su tamaño también es menor que el de dichos nodos. Esto es así porque el olor que contiene es más débil.

- Paso 3:
  - Siguiente nodo: *11*
  - $O(11) = O(4)\text{-}k \cdot w_{4,11} = 4\text{-}1 \cdot 3 = 1$ (al igual que ocurría con el nodo anterior, éste también se agrega a s_1)
  - $s_1 = \{1, 2, 5, 8, 9, 10, 4, 11\}$. En la Figura 3.8 (b) se puede ver como el nodo *11* aparece coloreado y con un tamaño de círculo mayor. No obstante el tamaño es menor que los otros nodos de s_1, y su color es más débil, esto es así porque ambas cosas hacen referencia al olor que contiene, que es menor que el de los otros elementos de s_1.

De modo que al final de la búsqueda, $|s_1|$ aumenta en dos unidades (pues ha incluido a dos nodos), y la siguiente vez que se busque un camino será suficiente con encontrar *11* en lugar de tener que ir hasta *5*.



(a) (b)

Figura 3.8. Inclusión de nuevos elementos en s_i.

3.5 Metodología

Antes de explicar la forma en la que se ha ido modificando el algoritmo ACO para satisfacer los objetivos marcados inicialmente, es necesario explicar la metodología llevada a cabo.

En concreto, en este caso se ha seguido un método de desarrollo iterativo incremental en el que cada ciclo incluye las fases mostradas en la Figura 3.9. Es decir, se va a seguir un modelo en espiral en el que cada ciclo incluye como objetivos el resolver los problemas encontrados en el ciclo anterior así como algunos nuevos para alcanzar un prototipo final que satisfaga los objetivos indicados en la introducción del presente trabajo.

Con respecto a las fases de cada ciclo, se pasan a explicar en detalle a continuación:

- Planteamiento (Fase 1 en la Figura 3.9): se definen los recursos máquina que van a ser necesarios para el ciclo actual así como los plazos temporales para finalizarlo.

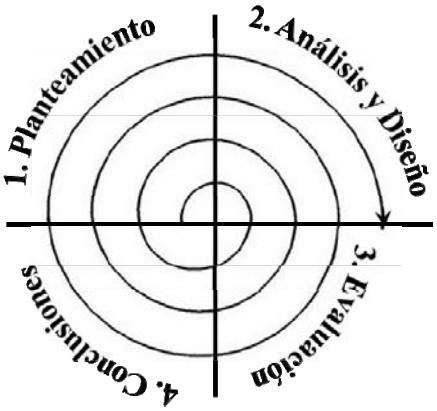


Figura 3.9. Desarrollo con ciclo en espiral.

- Análisis y Diseño (Fase 2 en la Figura 3.9): se definen los objetivos a perseguir en la fase actual (parte de los cuales implican resolver los problemas encontrados durante el ciclo anterior) y la manera en la que serán resueltos, es decir, se plantearán las nuevas modificaciones que se introducirán al algoritmo.
- Evaluación (Fase 3 en la Figura 3.9): una vez desarrollado la nueva versión del algoritmo, se pasará a una fase de evaluación en la que se medirá la bondad de las innovaciones introducidas, y se extraerán resultados de los parámetros de estudios relevantes en cada ciclo.
- Conclusiones (Fase 4 en la Figura 3.9): como fase final de cada ciclo, se extraerán una serie de conclusiones de la evaluación llevada a cabo, de manera que se comprobará si se han cumplido los objetivos marcados, y se determinará si hay algún aspecto que necesite ser mejorado en el siguiente ciclo.

Para explicar el contenido relativo a los ciclos del trabajo, se podría pensar en que cada uno de los siguientes capítulos fuese una iteración, con las cuatro fases que la componen, con el fin de hilar una con otra. No obstante, se ha decidido emplear otra estructura argumental para que el seguimiento del documento fuese más sencillo de cara al lector. Dicha estructura consiste en separar los conceptos teóricos de la evolución, de manera que primero se detalla toda la teoría asociada a cada ciclo, siguiendo esta explicación el proceso iterativo pero solo con las dos primeras fases, y posteriormente se estudia la evaluación realizada para cada una y la extracción de las conclusiones pertinentes, es decir, se siguen los ciclos iterativos pero solo teniendo en cuenta las fases tercera y cuarta.

Por lo tanto, en el Capítulo 4 se incluirán las fases primera y segunda de cada ciclo de vida, y en el Capítulo 5 la información correspondiente a las otras dos fases (tercera y cuarta). De esta manera, el planteamiento del problema así como el análisis y diseño del primer ciclo de evolución se encuentran en el apartado 4.1.1, en el 4.1.2 se muestra esa misma información pero relativa al segundo ciclo de evolución, la primera y segunda fase del tercer ciclo se muestran en

el 4.1.3, y en el apartado 4.2 se muestran las dos primeras fases del último ciclo de evolución de la propuesta. En lo que respecta a la evaluación correspondiente a cada uno de los ciclos, y de la cual se extraen una serie de conclusiones que permiten llevar al planteamiento del siguiente ciclo, se muestra en los apartados 5.3.1 (primer ciclo), 5.3.2 (segundo ciclo), 5.3.3 (tercer ciclo) y 5.3.4 (cuarto y último ciclo).

Capítulo 4 Evolución de la Propuesta

Para alcanzar los objetivos expuestos inicialmente relacionados con el algoritmo de búsqueda se realizaron una serie de versiones, cada una correspondiente a un ciclo de la metodología expuesta, en las que se fueron viendo las ventajas y desventajas de las modificaciones realizadas en el funcionamiento global del algoritmo teniendo en cuenta los tres principales parámetros de estudio:

- Tiempo de respuesta.
- Tasa de éxito en la búsqueda del camino.
- Calidad del camino encontrado.

Con cada versión se pretendía abarcar un nuevo requisito del total a alcanzar, de manera que la complejidad del problema a resolver en cada caso era creciente. No obstante, el hecho de incluir complejidad de manera paulatina implicaba que los problemas del ciclo anterior quedaban estudiados y resueltos, y no tenían que ser considerados en el ciclo siguiente, haciendo más fácil la tarea de conseguir el objetivo final.

Las diferentes versiones del algoritmo se dividieron en dos etapas principales: *algoritmo sobre un grafo estático* y *algoritmo sobre un grafo dinámico*, realizando en cada una de ellas una serie de experimentos que permitían comprobar si se estaban consiguiendo los objetivos que tuviesen asociados de manera individual las distintas etapas.

La razón de aplicar el algoritmo en primer lugar a grafos estáticos era conseguir que ACO funcionase adaptado a grafos vastos, es decir, ampliar el dominio de aplicación de las hormigas a grafos de centenares de miles de nodos, y además ajustar el valor de los principales parámetros que influyen en el buen funcionamiento del algoritmo así como incorporar distintas modificaciones al mismo. El trabajar sobre grafos estáticos facilitaba ambas cosas, pues era más sencillo crear un escenario de pruebas, y permitía obtener resultados más rápidamente que trabajando directamente sobre un grafo dinámico. De esta manera, una vez obtenido un

algoritmo robusto, con todos sus parámetros fijados, y con una total validez para trabajar sobre grafos vastos, se pasará a la siguiente etapa: *grafos dinámicos*.

Cada una de las etapas principales está dividida en distintas subetapas con el fin de ir llevando a cabo las distintas modificaciones adaptadas a las diferentes necesidades que se fueron detectando, pudiéndose identificar cada una de estas subetapas como ciclos dentro de la metodología seguida (apartado 3.5).

4.1 Grafo Vasto Estático

En esta etapa se trabajará con un grafo *G* sin cambios para cualquier valor de *T*, es decir, se trabajará con un grafo estático. El desarrollo de esta parte comenzará por resolver servicios sobre un grafo que tiene un único nodo comida, para generalizar después el problema a *n* nodos comida.

4.1.1 Un Nodo Comida

En esta primera fase de la experimentación se pretende comprobar si el algoritmo *SoSACO* es válido en grafos vastos, y para ello se empezará con la elección de un único nodo comida para observar los efectos que se producen en los resultados obtenidos.

Además de esto, se intentarán fijar valores para los distintos parámetros que forman parte de *SoSACO* de una manera rápida debido al hecho de no incluir dinamismo, lo que a su vez implica que no sea posible hacerlo para todos los parámetros, porque hay algunos de ellos que solo aparecen cuando se incluye esta característica en el grafo. En cuanto al análisis de los resultados, se tendrá en cuenta que en ciclos posteriores se incluirá dinamismo, de modo que en caso de que en estático fuese igual utilizar un valor u otro, se elegirá aquel que mejores resultados pueda aportar en dichos ciclos.

Aclarado esto, se van a pasar a explicar las modificaciones realizadas sobre ACO para hacerlo válido en grafos vastos debido a la incorporación de un nodo comida (f_1) y su olor (s_1), y cómo el método de búsqueda puede beneficiarse de este olor.

El algoritmo que describe el recorrido de cada hormiga de modo individual en esta primera versión, es el mostrado en Algoritmo 4.1.

Como se puede apreciar, el primer paso que es necesario realizar es vaciar la *lista tabú* que tiene asociada cada hormiga. El cómo utilizar esta *lista tabú*, y si utilizarla o no, va a ser uno de los primeros elementos a evaluar en la fase de experimentación.

Una vez se ha inicializado la *lista tabú* (*líneas 1 y 2* del Algoritmo 4.1) se pasaría a la búsqueda del camino desde el nodo inicio indicado (*start Node* en Algoritmo 4.1) al nodo destino (*end Node* en Algoritmo 4.1).

```
 1:  tabuList = empty;
 2:  tabuList.add(startNode);
 3:  node = startNode;
 4:  while (executionTime < t_threshold) do
 5:    if (node == endNode) then
 6:      tabuList.show();
 7:      tabuList = empty;
 8:      node = startNode;
 9:      tabuList.add(node);
10:    else
11:      odorAux = odor(node);
12:      if (odorAux > 0) then
13:        completePathThroughFood(node);
14:      else
15:        node = nextNodeSelection(node);
16:      end if;
17:    end if;
18:  end while;
```

Algoritmo 4.1. Búsqueda de caminos.

El identificador asociado a *start Node* y *end Node* no va a ser el mismo para todas las hormigas, ni va a ser igual en todos los casos. La forma de asignar valor a estos parámetros es la siguiente:

- Si alguno de los nodos inicio/destino del camino es f_1, todas las hormigas tendrán a dicho nodo como *end Node*, y como *start Node* al nodo contrario. El motivo de esta elección es la facilidad con la que un nodo comida es alcanzado desde cualquier nodo del grafo debido a la dispersión de olor realizada a su alrededor (al subconjunto s_i asociado a cada f_i). De manera que el tiempo de respuesta será menor realizando esta asignación.
- Si ninguno de los nodos inicio/destino es f_1, la mitad de las hormigas tendrán como *start Node* un extremo del camino y como *end Node* el contrario, mientras que el resto de hormigas (la mitad restante) escogerá el sentido contrario de búsqueda.

El comportamiento de estas dos colonias puede no ser el mismo. En la fase de evaluación se realizará un estudio comparativo para determinar su comportamiento.

Es importante indicar que el tiempo durante el cual se va a estar realizando la búsqueda va a estar limitado por el parámetro mencionado en el apartado de formalización $t_{threshold}$, y que es fijado por el analista, de manera que mientras que el tiempo de ejecución sea menor que dicho valor, los pasos que se van a seguir serán los mostrados en el Algoritmo 4.1 (*líneas 5-18*).

En ellos, en primer lugar, si el nodo en el que se encuentra la hormiga en ese momento es su *end Node*, se llamará al método *tabu List.show()* para mostrar el camino en caso de ser posible (antes de mostrarlo elimina todos los bucles que pudiesen haber surgido al realizar la búsqueda del camino). Como se podrá ver más tarde, es posible que la búsqueda se realice con una división en subcolonias. En caso de ocurrir esto, es posible que el nodo destino de la hormiga no sea un extremo del camino solicitado (esto se explicará en detalle posteriormente), de modo que cuando el nodo encontrado sea el destino de la hormiga, pero distinto a los nodos inicio/destino del camino solicitado, este método será el encargado de comprobar si existe el camino parcial

que lleva de ese nodo intermedio al nodo destino (es decir, comprobar si otra hormiga encontró ese camino parcial). Además de lo anteriormente mencionado, también tendrá que realizar la actualización global de la feromona en los casos que sea necesario y siguiendo las pautas fijadas por el tipo de algoritmo que se esté utilizando en ese momento. Una vez se ha hecho esto, la hormiga se situará en su nodo inicio y empezará de nuevo la búsqueda (lo que implica que la *lista tabú* solo contendrá ese nodo inicio).

```
1:  tabuListWithPathInsideOdor = tabuList + searchInsideOdor(node);
2:  oppositePath = oppositePathFromOdor(node,startnode);
3:  if (∃ oppositePath) then
4:    path = tabuListWithPathInsideOdor + oppositePath;
5:    path.eliminateLoops();
6:    path.showPath();
7:  end if;
8:  storePath(tabuListWithPathInsideOdor);
9:  node = nextNodeSelection(node);
```

Algoritmo 4.2. Búsqueda de caminos a través de s_i.

En caso de encontrarse en un nodo con olor distinto a *end Node*, si la cantidad de olor obtenido en *odor (node)* es mayor que cero, se llamará al método *complete Path Through Food (node)* para completar el camino que vaya desde el nodo con olor encontrado hasta f_1. Además, este método comprobará si es posible obtener un camino global hasta *end Node*. Esto último solo sería necesario en el caso de que *end Node* no sea f_1, pues de no ser así solo se necesitaría completar la *lista tabú* con el camino que va desde el nodo con olor encontrado hasta el nodo comida del mismo. En el Algoritmo 4.2 se puede ver de una forma más detallada los distintos pasos que se dan dentro del método *complete Path Through Food (node)*. Como se puede observar, un primer paso es buscar dentro del área con olor el camino que lleva desde el nodo con olor encontrado hasta f_1 (*search Inside Odor (node)*). La forma de realizar esta búsqueda será siguiendo el rastro de olor creciente. Una vez se ha completado la *lista tabú* con el camino hasta el nodo centro de olor, se pasa a comprobar si en algún momento anterior alguna hormiga encontró el camino que va desde el *end Node* de la hormiga en estudio hasta f_1 (método *opposite Path From Odor (node, start node)*). En caso de ser así, se unen los dos caminos parciales para obtener el global (el que va del *start Node* al *end Node*), se eliminan los posibles bucles que pudiesen resultar de la unión de los dos caminos parciales, y se muestra el camino obtenido con *path.showPath()*, que a su vez decidirá si realizar la actualización global de feromona o no en función de la versión de algoritmo empleada. A continuación, siempre se llamará al método *store Path (tabuListWithPathInsideOdor)* para almacenar el camino parcial encontrado desde el nodo inicio de la hormiga a f_1. Esto se hará si es la primera vez que se encuentra un camino con esos nodos extremos, o si es de menor coste que el anteriormente almacenado (borrando el camino que se tenía guardado). El fin de este almacenamiento es ayudar a encontrar el camino global, mediante la unión de caminos parciales, a hormigas futuras que alcancen s_1 y que lleven el sentido inverso. Una vez hecho esto, se llamará al método *next Node Selection (node)* que,

dependiendo de la versión del algoritmo empleada, reiniciará la búsqueda o no en función de la cantidad de olor existente en el nodo con olor encontrado, y elegirá el siguiente nodo al que moverse de una forma u otra. Además, cuando se realice un tipo de actualización de feromona local, es decir, que se actualice la feromona de cada enlace que se va visitando en cada momento, será este método el encargado de hacerlo.

Por último, si no se satisface ninguna de las condiciones anteriores, se elegirá el siguiente nodo al que moverse. La forma de realizar dicho movimiento vendrá dada por el método *next Node Selection (node)* que, como se ha indicado al final del párrafo anterior, tendrá una forma u otra de hacerlo en función del algoritmo concreto utilizado.

A continuación se va a proceder a explicar los tipos de actualización de feromona que se contemplan, las distintas formas de proceder cuando se encuentra un camino pasando por el olor, las formas de utilización de la *lista tabú*, así como la utilización de las distintas colonias para encontrar caminos.

4.1.1.1 Tipos de Actualización de la Feromona

La cantidad de feromona que contendrá cada enlace del grafo variará con el tiempo, ya que habrá determinadas situaciones que desencadenarán el cambio del valor asignado. Dichas situaciones podrán ser de dos tipos: que la hormiga se traslade a un nodo y que la hormiga haya encontrado un camino desde su nodo inicio a su nodo destino.

En el primero de los casos, la hormiga se traslada a un nodo, lo que se hará será realizar una actualización local. Su objetivo es actualizar la cantidad de feromona que hay en cada enlace utilizado en el momento en el que se utilizó. El nuevo valor de feromona asignado seguirá la Ecuación 4.1 en la que se puede apreciar que solo existe dependencia con el coste del enlace a actualizar, w_{ij}, y con la cantidad de feromona que existía previamente en dicho enlace, τ_{ij}. Este tipo de actualización se llevaría a cabo dentro del método *next Node Selection (node)* (*línea 15* del Algoritmo 4.1 o *línea 9* del Algoritmo 4.2).

$$\tau_{ij}(t) = \tau_{ij}(t-1) + \frac{cte}{1000 \cdot w_{ij}}$$

Ecuación 4.1. Actualización local.

Para el segundo de los casos, la hormiga ha encontrado un camino que va desde su nodo inicio hasta su nodo destino, se realizará una actualización global en todos los enlaces que forman parte del camino encontrado, de modo que la feromona cambia siguiendo la Ecuación 4.2. La diferencia que se observa en esta fórmula con respecto a la Ecuación 4.1 es que su influencia en el rastro de feromona es mucho mayor. Esto es así porque, tal y como se puede apreciar en la Ecuación 4.2, el término que se suma a la feromona anterior está dividido por la longitud del camino encontrado, que tendrá normalmente un valor inferior que el de $1000 \cdot w_{ij}$, es

decir, el incremento de feromona que se lleva a cabo en la actualización global es mayor que el llevado a cabo en la local.

$$\tau_{ij}(t) = \tau_{ij}(t-1) + \frac{cte}{pathLength}$$

Ecuación 4.2. Actualización global.

Cuando una hormiga encuentra un camino, antes de realizar el incremento mostrado en la Ecuación 4.2, disminuirá el valor previo de la feromona siguiendo la Ecuación 4.3, donde ρ es la tasa de evaporación y tomará valores entre 0 y 1. Dicha disminución se hará en la feromona de todos los enlaces del grafo, no como en el caso de la Ecuación 4.2 que solo se aplicaba a los enlaces que formaban parte del camino hallado.

$$\tau_{ij}(t) = (1-\rho) \cdot \tau_{ij}(t-1)$$

Ecuación 4.3. Evaporación de feromona.

En los casos en los que se lleva a cabo una actualización global se deberá tener en cuenta si el camino utiliza el olor o no. De manera que habrá dos tipos de comportamientos a tener en cuenta: cuando se encuentra el camino desde el nodo inicio del camino hasta el nodo destino del camino utilizando el olor a comida no se realizará la actualización global (ni la disminución ni el incremento), o bien, cuando ocurre esto se actualizará la feromona incrementándola en los enlaces del camino y disminuyéndola en todos los enlaces del grafo tal y como se hace cuando no se utiliza el olor para unir los dos nodos extremos del camino.

4.1.1.2 Actuación Cuando se Alcanza el Olor

Otro de los factores a tener en cuenta, y que propiciará el estudio de dos nuevas versiones del algoritmo, será el comportamiento de las hormigas después de utilizar el olor incluido en el método *next Node Selection (node)* del Algoritmo 4.2 (*línea 9*). El primero de los comportamientos se corresponde con que la hormiga vuelva a su nodo inicio y reinicie la búsqueda cuando el olor encontrado sea mayor o igual que *u*. Es decir, su proceso de búsqueda se reiniciará y empezará desde el principio cuando el nodo encontrado pertenezca al s_1 inicial. Esta reinicialización es debida a que los nodos pertenecientes a dicho subconjunto de s_1, a través de los rastros depositados en ellos, permiten obtener el camino para alcanzar f_1 de una manera óptima, mientras que aquellos con olor radial pueden conocer el camino óptimo o no (ya que depende de caminos anteriormente encontrados por otras hormigas). De esta manera, cuando el olor hallado es menor que *u*, es importante seguir buscando desde donde se encontró el nodo con olor para mejorar la calidad del camino hallado.

El segundo de los comportamientos se corresponde con hacer que la hormiga siga con la búsqueda de su nodo destino independientemente de la cantidad de olor que existiese en el nodo con olor alcanzado. Puesto que el objetivo de la hormiga es encontrar el camino desde su nodo inicio a su nodo destino, en este caso se seguirá con la búsqueda y no se reiniciará hasta que no

encuentre dicho objetivo. El hecho de encontrar un olor que permita construir un camino para alcanzar el destino es tomado en esta versión del algoritmo como una primera aproximación que se puede mejorar, y por lo tanto, aunque el camino encontrado pasando por el olor será mostrado como posible solución, la hormiga seguirá con su búsqueda desde el nodo con olor que se encontró para mejorar la calidad del mismo.

4.1.1.3 Utilización de la *Lista Tabú*

La utilización que se hace de la *lista tabú*, es decir, si se permite o no a las hormigas utilizar nodos anteriormente visitados, tiene una influencia directa en el funcionamiento del algoritmo. Con respecto a este tema, se optó por distintas opciones para afrontar el problema de la búsqueda.

La primera opción consistía en hacer una utilización estricta de la *Lista tabú*, de manera que se impidiese a las hormigas elegir nodos visitados. Con esto se garantizaba que en un mismo trayecto no se pasase más de una vez por un mismo nodo. No obstante, al ejecutar las pruebas de evaluación pertinentes se encontró que en un 90% de los casos las hormigas se perdían y no eran capaces de encontrar un camino. Además, en el 10% de los casos restantes, la calidad del camino encontrado, así como el tiempo de respuesta, se veían altamente perjudicados. El problema radicaba en que cuando había pasado gran cantidad de tiempo, las hormigas se encontraban con que no podían moverse a ningún nodo porque todos habían sido visitados con anterioridad. En este caso la única opción era reiniciarse y volver a empezar la búsqueda, enfrentándose al problema de la falta de tiempo para alcanzar el destino debido al tiempo consumido en la anterior búsqueda. Por todo esto se decidió estudiar controles distintos de la *lista tabú*.

La solución inicial consistió en no utilizar una *lista tabú*: para evitar que una hormiga llegue a un estado en el que no pueda avanzar porque todos los nodos adyacentes ya estén en la *lista tabú*, se elimina esta restricción. Esto trae consigo un problema directo: habrá nodos en el camino que se repetirán. Para tratar este problema, se incorporará un proceso encargado de eliminar bucles cuando se encuentre un camino desde el nodo inicio al nodo destino, consiguiendo que a lo largo del trayecto no se pase más de una vez por un mismo nodo. Es decir, en el método *show()* que se indica en la *línea 6* del Algoritmo 4.1, se eliminarán los bucles antes de mostrar el camino que ha seguido la hormiga para ir desde el nodo inicio al nodo destino.

En cuanto a la forma en la cual se decide el siguiente nodo al que ir cuando no se ha llegado al nodo destino de la hormiga ni al olor del nodo comida, es decir, lo que se hará en *next Node Selection (node)* (*línea 15* del Algoritmo 4.1), será una selección de un nodo de entre todos los adyacentes al nodo en el que se encuentra la hormiga basándose en la Ecuación 4.4 que tiene en

cuenta la feromona existente en los distintos enlaces que parten de él. Tal y como se puede apreciar en dicha fórmula, la probabilidad de ir de *i* (nodo actual) a *j* (nodo adyacente de *i*), donde $i, j \in N$, es igual a la cantidad de feromona asociada al enlace que los une, τ_{ij}, entre la suma de la cantidad de feromona de todos los enlaces que parten de *i* ($\sum_{x \in Adj} \tau_{ix}$ donde *Adj* es el conjunto de nodos adyacentes de *i* y τ_{ix} es la feromona asociada a cada uno de sus enlaces).

Esta selección de siguiente nodo se realizará siempre y cuando el tiempo de ejecución transcurrido no supere $t_{threshold}$.

$$p(i,j) = \frac{\tau_{ij}}{\sum_{x \in Adj} \tau_{ix}}$$

Ecuación 4.4. Probabilidad de elección del siguiente nodo.

Como se puede apreciar, se ha eliminado la parte heurística de la ecuación genérica de selección de siguiente nodo de ACO (Ecuación 2.1). Esto es así porque se quería buscar en grafos de los cuales no se tuviese información previa y en los cuales no se supiese dónde estaba el destino (por ejemplo en las redes sociales o en las redes de interacción de proteínas), de manera que no sea posible aplicar una heurística acertada.

Una solución menos drástica era incluir un control relajado de la *lista tabú* que se denominó *β-tabú*. Su fin es evitar la aparición de bucles en el camino pero de una forma más relajada que en el caso del control estricto de la *lista tabú*. El principio que sigue *β-tabú* es el siguiente: solo se evitará la utilización de nodos visitados con anterioridad en los pasos múltiplos de *β*. Para ello, si en ninguno de los *β-1* pasos anteriores se utilizó un nodo previamente visitado, en el paso múltiplo de *β* se podrá elegir cualquier nodo de entre todos los adyacentes (visitados con anterioridad o no), y en el caso en el que en alguno de los *β-1* pasos anteriores se haya utilizado un nodo visitado previamente, solo se podrá elegir entre los nodos adyacentes no visitados, si esto es posible, o entre todos si todos los nodos adyacentes son nodos visitados. Una vez se tiene el subconjunto de nodos de *N* entre los cuales elegir el siguiente nodo a visitar, la elección se realizará siguiendo la Ecuación 4.4, de modo que dicho subconjunto será el que incluya *Adj*.

Con el fin de hacer más sencilla la comprobación de si en alguno de los *β-1* pasos anteriores se empleó un nodo visitado o no, se utilizará un *flag* que será activado en el momento en que se elija un nodo que ya estuviese en la *lista tabú*.

Respecto al caso especial de $\beta = 1$ lo que se tendrá es que siempre se tratará de ir a aquellos nodos que no se emplearon con anterioridad. Es decir, se considerará que el *flag* siempre está activo.

Pues bien, todo este proceso será el llevado a cabo en *next Node Selection (node)* (*línea 15* del Algoritmo 4.1) mientras que el tiempo de ejecución sea menor que $t_{threshold}$.

Además, al igual que ocurría cuando no se utiliza *lista tabú*, como habrá ocasiones en las que se tenga que un nodo haya sido utilizado en repetidas ocasiones dentro del mismo camino, podrán existir bucles. Por este motivo, dentro de *show ()* (*línea 6* del Algoritmo 4.1), y antes de mostrar el camino obtenido, se eliminarán todos los bucles que puedan existir.

4.1.1.4 División en Distintas Colonias

Debido a las características del problema al cual se enfrenta el algoritmo propuesto, *SoSACO*, es importante realizar una correcta elección de la forma en la cual se van a distribuir las hormigas dentro de todo el grafo, ya que, en parte, el buen funcionamiento del algoritmo va a depender de esta distribución.

Para encontrar la solución más adecuada se ha probado con distintos tipos de distribución, de manera que tras la posterior evaluación se elegirá la que ofrezca unos mejores resultados (aquella que permita una búsqueda más eficiente y eficaz).

Es importante destacar que, en cualquiera de los casos de distribución de las hormigas que se explicarán a continuación, cuando uno de los nodos extremo del camino sea f_1, todas las hormigas se situarán en el otro nodo extremo del camino y su misión será encontrar f_1. Esto es así porque debido a la dispersión de olor que se realiza en torno a f_1 será muy sencillo encontrar el nodo centro de dicha dispersión, y por lo tanto, si hay un mayor número de hormigas tratando de encontrar un rastro de olor, el tiempo de respuesta se reducirá. A esto, hay que añadir el hecho de que buscar el camino inverso (hormigas que parten de f_1 y buscan el otro nodo extremo) llevaría más tiempo, e incluso habría casos en los que fuese imposible alcanzar una solución, ya que s_1 sería omitida por no aportar ninguna información, de modo que sería como ejecutar un algoritmo básico de ACO sobre un grafo vasto sin ningún tipo de ayuda.

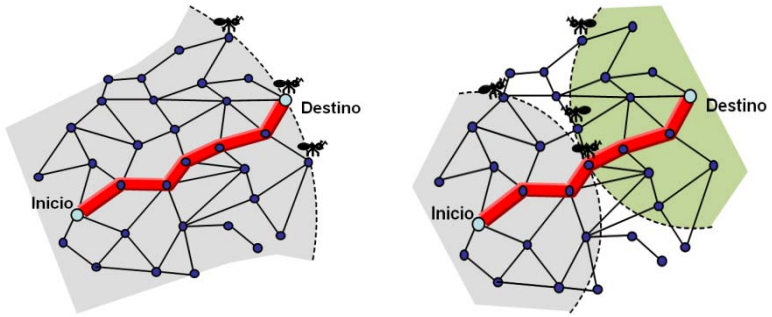
Las distintas alternativas que se plantearon fueron las que se denominaron Divide y Vencerás Simple y Divide y Vencerás con Subdivisión.

En el tipo Divide y Vencerás Simple, lo que se hará será crear dos colonias de hormigas de modo que cada una de ellas tendrá como inicio uno de los nodos extremo del camino solicitado, y como destino el nodo contrario. Además, cada colonia tendrá asignado un tipo propio de feromona. Esto implicará que a la hora de dejar un rastro de feromona cada colonia depositará un tipo distinto a cada paso que dé en caso de existir actualización de feromona local (la colonia *A* depositará feromona de tipo *A*, y la colonia *B* depositará feromona de tipo *B*), y que si se realiza actualización global ambas colonias actualizarían la feromona tanto del tipo *A* como del tipo *B* que se encuentre depositada en cada enlace del camino.

En lo referente a la forma de seguir el rastro, las hormigas no buscarán su propio tipo de feromona, lo que harán será intentar moverse en función del rastro de feromona de la otra colonia (siguiendo la Ecuación 4.4 mostrada anteriormente), de manera que si en un paso no se

encuentra ningún rastro de feromona de la otra colonia, la hormiga elegirá de modo aleatorio uno de los nodos accesibles desde el lugar en el que se encuentre.

El dejar un rastro y seguir otro se basa en que se quieren dar pistas a las otras hormigas que están buscando, ya que no interesa buscar el hormiguero propio, sino que interesa buscar el hormiguero de la otra colonia (pues es el nodo destino de las hormigas de la primera colonia mencionada). Al seguir el rastro contrario, lo que se consigue es ir hacia aquella zona en la que abunda dicha feromona, que suele ser en los alrededores del hormiguero (ya que luego las hormigas se empiezan a dispersar), es decir, en los alrededores del nodo destino de la hormiga. De manera que completar el camino una vez que se ha encontrado el rastro de la colonia opuesta es mucho más rápido.



(a) (b)

Figura 4.1. Ejemplo de división de colonias y actuación cuando se alcanza la feromona opuesta.

Esta característica supone en sí misma una mejora con respecto al algoritmo ACO sin añadir nada relativo al olor, ya que el tiempo que requieren *x* hormigas para cubrir un círculo de radio *r* igual a la longitud del camino solicitado es mayor que el requerido para cubrir uno de radio *r/2*.

Un ejemplo de esto se puede apreciar en la Figura 4.1, en la que se observa como cuando todas las hormigas parten del mismo nodo, Figura 4.1 (a), tienen que recorrer todo el camino para dar una solución, mientras que en el caso de que las hormigas usen Divide y Vencerás Simple, Figura 4.1 (b), cuando se encuentra la feromona de la colonia opuesta (el círculo cubierto por el tipo de feromona gris toca al círculo del tipo de feromona verde) será más sencillo encontrar un camino global, pues solo habrá que seguir el rastro creciente de dicha feromona opuesta.

Con respecto al tipo Divide y Vencerás con Subdivisión, el proceso de búsqueda es más complejo que el explicado anteriormente, aunque también se basa en la división en subcolonias de la colonia de hormigas inicial. En este caso, la colonia inicial de hormigas se divide en dos colonias: una (*A*) tendrá como hormiguero el nodo inicio del camino y como nodo destino el nodo destino del camino solicitado, y la otra colonia (*B*) tendrá como hormiguero el nodo destino del camino y como nodo destino el nodo inicio del camino solicitado.

Las hormigas de la colonia *A* tendrán feromona del tipo *A* y las de la colonia *B* tendrán feromona de tipo *B*. De manera que al igual que pasaba en el caso anterior, la colonia *A* buscará la feromona de tipo *B* y depositará feromona de tipo *A* en cada paso que dé, en caso de existir actualización local, y la colonia *B* actuará al contrario (buscará feromona de tipo *A* y depositará feromona de tipo *B* en cada enlace que visite cuando exista actualización local). En el caso de la actualización global, ambas colonias depositarán la feromona de tipo *A* y *B*.

Cuando se encuentra el primer enlace con resto de feromona de la colonia opuesta, es decir, cuando se produce el choque de los dos círculos de feromona representado en la Figura 4.1 (b), la colonia que lo encuentra sufre una división (solo se realizará esta división en una ocasión): parte de la colonia tendrá como nodo inicio el que tenía anteriormente y como nodo destino el nodo en el cual hay un enlace con feromona de la otra colonia, que se pasará a denominarse nodo intermedio, y otra parte de la colonia pasará a tener como nodo inicio dicho nodo, y como nodo destino el nodo destino inicial. El número de hormigas que forma parte de cada nueva colonia vendrá fijado por un parámetro (η) que determinará el analista. El objetivo principal de esta nueva subdivisión es obtener un camino de la forma más rápida posible, de manera que una vez que se localiza la feromona de la colonia opuesta, las hormigas saben cómo encontrar el hormiguero de dicha colonia. Además de esto, el crear una nueva colonia permitirá mejorar el coste del camino encontrado, ya que se mejorará el coste del camino parcial que va desde cada hormiguero a sus nuevos nodos destino.

Debido a la aparición de una nueva colonia, también es necesario un nuevo tipo de feromona que tendrá que ver con ella. Dicho tipo de feromona será el identificado como *C*. A la hora de dejar el rastro, en el caso de que se utilice actualización local, la hormiga actualizará la feromona de tipo *A* si su hormiguero es el nodo inicio del camino. En caso de que el nodo inicio de la colonia a la que pertenece la hormiga sea el nodo destino del camino, la feromona actualizada en el enlace será de tipo *B*. Por el contrario, si la hormiga pertenece a la nueva colonia de hormigas creada, es decir, a aquella que tiene como nodo inicio el nodo intermedio del camino, el tipo de feromona que se depositará será el *C*.

Si se realizase una actualización global, el tipo de feromona depositada sería el que tenga asignado su colonia así como el de la colonia que vaya en sentido contrario: si el nodo inicio/destino es el nodo inicio del camino solicitado, y el nodo destino/inicio es el nodo destino de dicho camino, se actualizará la feromona de tipo *A* y *B*; si el nodo inicio/destino es el nodo inicio del camino y el nodo destino/inicio es el nodo intermedio seleccionado, la feromona que se actualizará será la de tipo *A* y *C*; en caso de que el nodo inicio/destino sea el nodo destino del camino solicitado, y de que el nodo destino/inicio sea el nodo intermedio, se actualizará la feromona de tipo *B* y de tipo *C*.

A partir del momento de la división, la feromona en la que se fijará cada colonia para elegir el siguiente nodo al que moverse será aquel tipo de feromona que pertenezca a la colonia que tiene como nodo inicio su nodo destino: la colonia que tenga como hormiguero el nodo inicio del camino y como destino el nodo destino del mismo se basará en el tipo *B* de feromona para elegir el siguiente nodo al que moverse; la colonia que vaya en sentido contrario a la anterior (del nodo destino del camino al nodo inicio del camino) se basará en la feromona de tipo *A*; la colonia que vaya desde el nodo inicio/destino al nodo intermedio encontrado realizará su elección teniendo en cuenta la feromona de tipo *C*; la colonia que vaya del nodo intermedio al nodo inicio del camino utilizará la feromona de tipo *A*; por último, aquellas hormigas que vayan del nodo intermedio al nodo destino del camino utilizarán el depósito de feromona de tipo *B* que exista en los enlaces que parten del nodo en el que se encuentra.

En este tipo de división, el camino estará completo cuando se encuentre uno de los extremos del camino partiendo del otro nodo extremo, o cuando se encuentre un camino global en base a la unión de los caminos parciales (nodo inicio/destino - nodo intermedio) + (nodo intermedio - nodo destino/inicio) pasando o no por el s_1. Además, al igual que se almacena el mejor camino parcial que une cada nodo extremo del camino con el nodo comida, también se guardará el mejor camino parcial que une cada uno de los nodos extremo con el nodo intermedio, y el que une el nodo intermedio con el nodo comida.

4.1.2 Varios Nodos Comida

Puesto que, tal y como se explicó en el apartado 3.3, el hecho de tener valores de *u* altos, áreas de olor entorno a los nodos comida pequeños, implicaba una rápida adaptación a cambios (pues la difusión del olor se realiza en menos tiempo) y una baja probabilidad de que los mismos le afectasen, sería interesante tomar como siguiente ciclo en la evolución del algoritmo el ubicar distintos nodos comida con áreas de olor pequeñas por todo el grafo para aprovechar esas ventajas, y compensar la dificultad de encontrar el área de olor debido a su tamaño con la cantidad de las mismas (aumenta la probabilidad de tener una cerca de nodo inicio).

Por todo ello, una vez se tiene una primera versión del algoritmo, se va a proceder en este apartado a aplicar el algoritmo a un escenario en el cual existe más de un nodo comida, y por tanto se explicarán todas las alternativas planteadas para adaptarlo y que resulte eficiente en este nuevo contexto.

En este caso, al igual que se hacía en el caso anterior, se van a almacenar los caminos encontrados por las hormigas para ir desde su nodo inicio al nodo comida. La única diferencia con respecto al caso anterior, es que ahora habrá muchos caminos parciales, ya que podría darse el caso de que existiese un camino almacenado por cada nodo comida y nodo extremo del camino. Esto supondría gran cantidad de información a almacenar. No obstante, el hecho de

utilizar una base de datos va a ahorrar los problemas que pudieran surgir del manejo de esta cantidad masiva de datos, proporcionando por añadidura todos los mecanismos necesarios para garantizar la concurrencia, manejo de memorias intermedias, escalabilidad, etc.

Las principales innovaciones que se añadirán al algoritmo obtenido al final del apartado anterior tendrán que ver con el hecho de incorporar, o no, una fase en la cual se compruebe si el olor a comida detectado ha sido utilizado anteriormente, y si es conveniente utilizarlo de nuevo o no. Además, será importante determinar la semántica asociada a *u* cuando existen varios nodos comidas.

4.1.2.1 Comprobación de Olor Encontrado

Cuando en un grafo existen varios nodos comida con sus correspondientes áreas de olor s_i, es importante considerar si es necesario incluir una fase en la que se compruebe si el olor encontrado debe tenerse en cuenta o no, para estudiar si se puede crear un camino que de solución al servicio solicitado a partir de él.

La primera de las alternativas es realizar siempre una comprobación del olor. Es decir, cada vez que se visite un nodo que posea un determinado rastro de olor, se comprobará si existe un camino o no a través de él mediante la unión del camino parcial que se encontró y otro encontrado por otra hormiga. Después de esto, la hormiga proseguirá o no con su búsqueda en función de la decisión tomada de la experimentación correspondiente al apartado 4.1.1.

La otra opción es restringir el número de comprobaciones de manera que solo se comprobará si se puede o no conseguir un camino solución del servicio solicitado utilizando el nodo comida cuyo rastro de olor se ha encontrado cuando dicho tipo de olor no haya sido visitado anteriormente por la hormiga con un olor mayor o igual que *u*, o cuando simplemente no se haya visitado nunca con anterioridad. Es decir, cuando una hormiga encuentra un s_i, lo que hará en primer lugar será comprobar que ese olor no esté marcado como visitado. Si está marcado, no realizará ningún tipo de comprobación, y si no lo está, mirará si la cantidad de olor encontrada es mayor o igual que *u*. En caso de serlo, marcará el tipo de olor como visitado, de manera que en posteriores visitas no se realizarán cálculos con él, y si es menor que *u*, no se marcará el nodo como visitado, pues el camino desde el nodo con dicha cantidad de olor hasta el nodo comida no tiene porque ser el óptimo, permitiendo que se puedan seguir realizando comprobaciones futuras con dicho tipo de olor y mejorar el camino. Una vez realizado esto, comprobará si puede obtener un camino que vaya del nodo inicio del camino al nodo destino empleando el olor encontrado. Este proceso intenta ahorrar tiempo de respuesta disminuyendo la cantidad de proceso.

Se podría pensar que con la segunda alternativa no se realizarían comprobaciones, por estar el olor encontrado marcado, en las que los resultados podrían ser satisfactorios porque desde la

otra colonia (por ejemplo *B*) se hubiese alcanzado ya el olor a comida, y por lo tanto sí que se podría construir un camino global. No obstante esto no es un problema, ya que los caminos parciales encontrados (los caminos entre los nodos inicio y los nodos comida) se almacenan cuando se encuentran, de manera que la hipotética colonia *B* ya habría mostrado el camino global con anterioridad.

4.1.2.2 Utilización del Parámetro *u*

Como se explicó en el apartado 3.3, el parámetro *u* determina el mínimo valor de olor que puede tener un s_i en el momento de la difusión del olor inicial. Esta definición era así cuando se tenía un único nodo comida. No obstante, es importante plantearse el modo de interpretarlo cuando existen varios nodos comida.

Una de las posibles interpretaciones es que el valor de *u* sea igual para todos los nodos comida, es decir, $\forall s_i \in S$, el valor de *u* será el mismo. Con esta opción se permite dar más peso a aquellos nodos que tengan un mayor grado dentro del grafo que a aquellos que lo tengan menor, ya que las áreas de los primeros serán mayores que las de los segundos. El principal problema de esta opción radica en el hecho de que cuando se quiera tener un área de olor grande entorno a los nodos comida con un grado menor, el área de los que tengan el grado mayor alcanzará grandes dimensiones, lo cual implica una probabilidad más alta de verse afectados por cambios, así como un aumento en la cantidad de tiempo que se necesita para regenerarlo.

Otra interpretación implicaría un valor de *u* distinto para cada nodo comida, es decir, $\forall s_i \in S$ existirá un u_i asociado, de manera que al conjunto de todos los posibles valores de *u* se le denominará $U = \{u_i\}$ y se cumplirá que $|U|=|S|$. Con esta opción se podría definir el tamaño de cada área de olor individualmente solventando el problema anteriormente mencionado. Esto permitiría, por ejemplo, dar el mismo peso a todos los nodos comida, de manera que no haya unos con áreas mayores que otros.

4.1.3 Varios Nodos Comida y Caminos Preprocesados Entre Ellos

El siguiente paso en la evolución de *SoSACO* implicaba una fase paralela a la de búsqueda del camino solicitado en la que se hallasen los caminos entre los distintos f_i con la expectativa de disminuir el tiempo de respuesta, pues al tener preprocesados los caminos que unen los nodos comida solo es necesario que una hormiga de una colonia alcance un olor y una hormiga de la colonia opuesta alcance otro, o el mismo, para tener una primera solución.

Por ejemplo, esto se puede ver en la Figura 4.2, en la que en el momento en que una hormiga ha alcanzado s_1 (en este caso a través de un radio de olor), y que una hormiga que parte del nodo extremo opuesto del camino ha llegado a s_2 (en este caso a través del olor dispersado inicialmente), se puede construir un camino global gracias a tener preprocesado el camino que

une f_1 y f_2. Otro ejemplo se muestra en la Figura 4.3, en la que se tendría el camino que une el nodo inicio y el nodo destino gracias a la concatenación de caminos entre distintos nodos comida (*Inicio* - f_9 - f_4 - f_5 - f_6 - f_7 - f_8 - *Destino*).

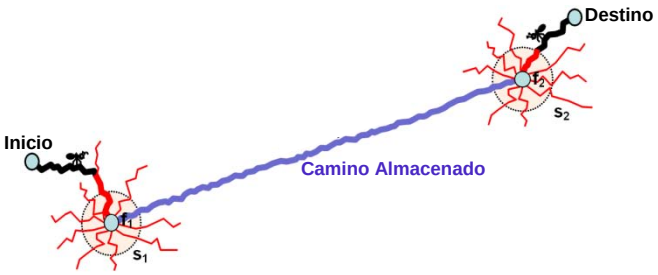


Figura 4.2. Camino global a través de caminos preprocesados.

Este procesamiento paralelo se llevaría a cabo, idealmente, en una máquina distinta a aquella en la que se ejecutan los servicios (con el fin de evitar efectos negativos en los tiempos de respuesta), y los métodos para realizar este preprocesamiento serían Dijkstra y ACO.

En caso de emplear Dijkstra, se calcularán todos los caminos entre los nodos comida que se hayan distribuido en el grafo y se almacenarán. A pesar de que esta opción calcula los caminos óptimos, tiene el problema de que si las variaciones que experimenta el grafo son frecuentes, podría ocurrir que uno de estos caminos se viese afectado y que nunca fuese posible recuperarse de dicha modificación, pues el camino obtenido se quedaría invalidado durante el proceso de búsqueda. Es decir, el lanzamiento de una búsqueda con el algoritmo de Dijkstra cuando existiese un cambio no mostraría ninguna alternativa válida al camino dañado.

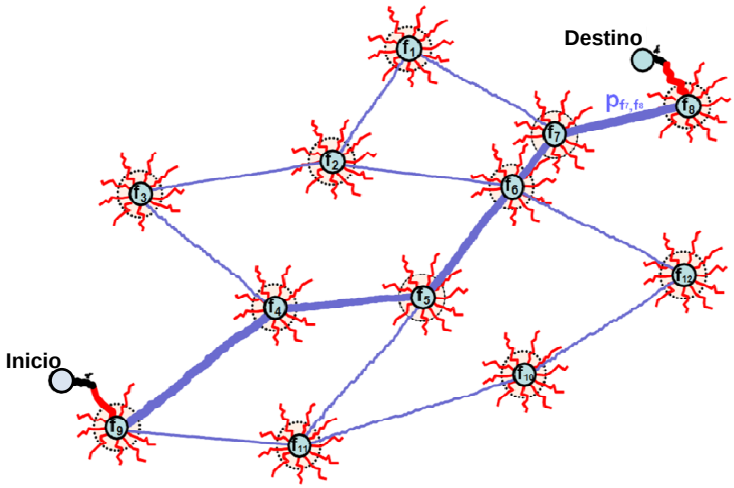


Figura 4.3. Varios caminos preprocesados y ejemplo de dinamismo en él.

La otra alternativa, emplear ACO, es una buena opción de cara al dinamismo expresado anteriormente por su adaptabilidad ante cambios. Este hecho permitiría que si uno de los caminos entre nodos comida se viese afectado por un cambio, las hormigas se lanzarían para intentar arreglarlo, y mostrarían una alternativa en poco tiempo. Por lo tanto, se dispondría de una fase previa en la que se calcularían los caminos, y posteriormente existiría una fase de mantenimiento de dichos caminos, que correría de manera paralela en otro servidor, para que

siempre se encontrasen en un estado consistente y para tratar de mejorar los caminos ya calculados, ya que si se encuentra uno mejor que el almacenado, se borrará el antiguo y se tendrá en cuenta el nuevo. Respecto al algoritmo ACO empleado, contaría con las mismas innovaciones que se emplearon con anterioridad con las siguientes diferencias: las hormigas se distribuirían de manera uniforme por todos los nodos comida y los nodos destino serían cualquier nodo comida, no uno en exclusividad, con el fin de encontrar lo más rápido posible caminos entre pares de ellos; la feromona depositada por estas hormigas no la verían las hormigas que resuelven servicios para evitar una interferencia negativa en el tiempo de respuesta de estas últimas; solo existiría actualización global y la cantidad depositada sería siempre igual a *1* con el fin de que las hormigas no se estabilizasen en los caminos que unen los nodos comida más cercanos entre sí, y que pudiesen seguir explorando para poder unir más nodos comida.

De este modo, si se dejase ejecutar el preprocesado con ACO durante gran cantidad de tiempo, se podrían tener los caminos entre cualesquiera dos nodos comida al igual que ocurría en el caso de emplear Dijkstra. No obstante, debido a la perspectiva futura de aplicar el algoritmo a grafos dinámicos, los caminos entre los nodos comida más alejados entre sí tendrían un tiempo de vida bajo, pues al poseer una mayor cantidad de elementos, tendrían una mayor probabilidad de verse afectados por los cambios, y por lo tanto de ser eliminados. Por ello, los caminos preprocesados que se mantendrían más estables con el tiempo serían los de los nodos comida cercanos por estar compuestos de menos elementos, y porque las hormigas podrán encontrar caminos válidos alternativos rápidamente, pues aunque el camino se borra no lo hace así la feromona depositada en los enlaces que lo forman. Por estos motivos, un ejemplo del resultado obtenido podría ser el mostrado en la Figura 4.3, en el que los caminos preprocesados que se mantienen son aquellos existentes entre los nodos comida próximos entre sí. El problema que podría plantear utilizar ACO, vendría de la baja calidad de los caminos obtenidos, sin embargo como la fase de cálculo de caminos entre nodos comida se realiza *off-line* empleando todo el tiempo que sea necesario, será posible la estabilización de ACO y por tanto, la obtención de calidades óptimas en los caminos calculados.

Por último, indicar que si el dinamismo es alto, podría ocurrir que algún nodo comida no estuviese conectado con el resto. No obstante, esto no afectaría a la búsqueda, porque en el caso de que una hormiga lo alcanzase y viese que no es posible construir un camino, seguiría buscando para encontrar su nodo destino u otro nodo comida. Por ejemplo, si en la Figura 4.3 se eliminase el camino que une f_7 con f_8, cuando la hormiga alcanzase f_8 no podría obtener un camino global, sin embargo, seguiría buscando y sería bastante probable que en poco tiempo más alcanzase f_7, por ser cercano a f_8, y obtendría un camino.

4.2 Grafo Vasto Dinámico

Una vez estudiados los distintos ciclos a los que se ha sometido al algoritmo *SoSACO* en estático, y después de introducir y explicar una serie de parámetros que afectan a su funcionamiento, en este ciclo se pasan a considerar aquellos aspectos relativos al dinamismo. Aunque antes de hacerlo, es necesario fijar la definición de dinamismo que se manejará en este trabajo de tesis doctoral. De modo que, puesto que en la mayoría de los trabajos existentes en la actualidad se tiene en cuenta como tipo de cambio sobre grafos aquel relativo a las variaciones en los costes de los enlaces, en este trabajo se entenderá por dinamicidad la aparición/desaparición de nodos y enlaces en el grafo. Es decir, se tendrá que con una periodicidad de tiempo *T*, previamente definida por el analista, aparecerá o desaparecerá un nodo/enlace.

Las principales características a estudiar del algoritmo *SoSACO* en este ciclo son las siguientes: efecto de los cambios en las hormigas, y comportamiento de las áreas de olor ante el dinamismo (aunque las ideas generales de este comportamiento se explicaron en el Capítulo 3, en esta sección se explicarán en detalle).

4.2.1 Efecto de los Cambios en las Hormigas

La desaparición de nodos y enlaces puede afectar a las hormigas que están buscando un camino en el momento de dicho cambio, pues si alguna hubiese empleado en su trayecto el elemento del grafo que va a desaparecer, mostraría una solución no válida al final de la búsqueda. Esto implica que el algoritmo deberá incorporar un mecanismo que evite esta situación. Para ello, la primera opción planteada podría ser matar a esas hormigas y reiniciarlas. Para llevar a cabo esto se tendrían que identificar todas las hormigas que están ejecutando en ese momento sin repercutir en el tiempo de respuesta. Este hecho no es viable, y por este motivo lo que se hará será dejar a todas las hormigas que estén ejecutando que prosigan con su búsqueda, y lanzar otras nuevas que suplan a las afectadas mientras que éstas sigan buscando el resto de un camino que no tendrá validez por contener un elemento que ya no existe. De esta manera, se evitará perder recursos de búsqueda, pues las nuevas hormigas comenzarán su ejecución sin tener que esperar a que las afectadas (denominadas hormigas *zombi*) terminen, manteniendo un número alto de hormigas válidas buscando caminos.

El número de hormigas nuevas que se lanzarán será directamente proporcional a la cantidad de feromona que dependa del elemento que desaparece del grafo, es decir, será igual a dicha cantidad de feromona multiplicada por un factor de ponderación *h* que deberá ser elegido por el analista. La cantidad de feromona a tener en cuenta para el cálculo podrá ser la que tenga asignada un enlace, en caso de ser un enlace el elemento que se pretende borrar, Ecuación 4.5

(b), o la suma de la feromona de todos los enlaces que parten del nodo que desaparece en caso de ser este tipo de elemento el que se borra del grafo (Ecuación 4.5 (a) donde *Adj(i)* representa a todos los nodos adyacentes al nodo *i* que se pretende borrar).

$$\#NuevasHormigas|_x = h \cdot \sum_{j \in Adj(i)} \tau_{ij}\Big|_x \quad \text{(a)}$$

$$\#NuevasHormigas|_x = h \cdot \tau_{ij}\Big|_x \quad \text{(b)}$$

Ecuación 4.5. Nuevas hormigas de tipo *x* cuando se elimina el nodo *i* (a) o el enlace l_{ij} (b).

El que el número de nuevas hormigas dependa de la feromona, va a permitir que si el rastro es muy débil (porque pasaron pocas hormigas, o porque las que pasaron lo hicieron hace mucho tiempo y la feromona se ha ido evaporando durante la ejecución del algoritmo) el número de nuevas hormigas sea bajo, y que si es fuerte (porque ese elemento haya sido empleado recientemente o por muchas hormigas) el número de nuevas hormigas sea elevado.

Como podrían existir tres tipos de feromona (debido al tipo de división de la colonia Divide y Vencerás con Subdivisión), se calculará el número de hormigas de la forma explicada anteriormente para cada uno de estos tipos, de manera que cada nueva hormiga tendrá el nodo inicio y destino que indique el tipo de feromona al que debe su existencia.

Por último, cuando una hormiga llegue a una condición de parada se comprobará si en su camino hay algún elemento inactivo, de manera que si no lo hay actuará como una hormiga normal (comportamiento ante una situación en la que se alcanzó una condición de parada explicada en ciclos anteriores), y si lo hay, si es una hormiga *zombi*, y el número de hormigas que hay corriendo en ese momento supera un determinado valor, morirá, pues se considerará que ya hay suficientes hormigas buscando una solución válida para el servicio solicitado.

4.2.2 Comportamiento de las Áreas de Olor frente a Cambios

Puesto que el algoritmo ACO en sí está adaptado para trabajar sobre entornos dinámicos (prueba de ello son los trabajos detallados en el apartado 2.3.2), la única parte de *SoSACO* que tiene que adaptarse para tratar con los problemas de grafos dinámicos es la relativa a las áreas de olor s_i.

Para conseguir que esta parte del algoritmo sea también adaptable a cambios, y que el algoritmo entero lo sea en todas sus partes por separado, se realizarán las siguientes acciones en caso de que un cambio afecte a un s_i:

- Si el cambio afecta a la zona de s_i con olor comprendido entre *m* y *u*, se disipará el olor de todos los nodos con un olor menor que el del elemento borrado (el olor del nodo borrado, o del nodo con menor olor del enlace que se ha borrado) y que hayan obtenido dicho olor del elemento a borrar. Una vez hecho esto, se difundirá el olor desde los

nodos que rodeaban al nodo borrado, o desde el nodo con menor olor del enlace borrado, mientras que el olor a asignar sea mayor o igual que la cantidad mínima de olor para ese tipo concreto de olor a comida. Es decir, se seguirá el mismo proceso que en la dispersión de olor inicial pero partiendo de nodos distintos a los nodos comida y con una cantidad de olor inicial a dispersar, generalmente, menor que *m*.

- Si el cambio afecta a la zona de s_i con olor menor que *u*, es decir, al olor radial, únicamente se realizará la fase explicada en el caso anterior en la que se borraba el olor de algunos nodos.

Nombre	Entradas	Descripción	Invocaciones a Métodos
link Status To False	Enlace eliminado	Selecciona el nodo extremo del enlace con menor olor y decide si regenerar el rastro de olor o simplemente borrarlo.	*regenerate Odor Link* (si se decide regenerar) o *delete Odor* (si se decide borrar)
regenerate Odor Link	Nodo desde el que dispersar olor	Borra el rastro de olor que parte del nodo de entrada	*diffusion*
link Status To True	Enlace creado	Comprueba si el olor de alguno de los nodos extremo del enlace de entrada es mayor que el valor de *u* del tipo de olor bajo estudio.	*Ninguno* (olor de los nodos extremo menor o igual que *u*) o *diffusion* (olor de alguno de los nodos extremo mayor que *u*)
regenerate Odor Node	Nodo borrado con olor mayor o igual que *u*	Elimina rastro de olor que parte del nodo de entrada.	*diffusion*
delete Odor	Nodo borrado con olor menor que *u*	Elimina rastro de olor que parte del nodo de entrada.	*Ninguno*
diffusion	Nodo	Dispersa el olor desde el nodo de entrada mientras que el olor a asignar sea mayor o igual que *u*.	*Ninguno*

Tabla 4.1. Tabla resumen de los métodos encargados de mantener en un estado consistente las s_i.

Con el fin de que este proceso quede más claro, a continuación se pasan a explicar en detalle los pasos a realizar en todas las situaciones posibles:

- Se elimina un nodo del grafo: se obtendrán todos los tipos de olor que tenga el nodo en cuestión, y por cada uno de ellos se analizará la cantidad de olor depositada en él, de manera que si es menor que el valor mínimo de olor fijado para ese tipo de olor pero mayor que cero, se ejecutará *delete Odor* (Figura 4.8), y si es mayor que dicho valor, se ejecutará *regenerate Odor Node* (Figura 4.7).

- Se crea un nodo: en este caso, se comprobará si es posible difundir algún tipo de olor desde el nuevo nodo, así como asignarle a él una cantidad del mismo. El método que se ejecutará para cada uno de los tipos de olor existentes será *diffusion* (Figura 4.9).
- Se elimina un enlace: cuando vaya a desaparecer un enlace del grafo se ejecutará *link Status To False* (Figura 4.4), pasándole como entrada el enlace a borrar l_{ij}. Este método se ejecutará por cada tipo de olor que exista en el grafo.
- Se crea un enlace: en este caso, se ejecutará *link Status To True* (Figura 4.6) por cada uno de los tipos de olor que tenga el grafo sobre el que se trabaja. A dicho método se le pasará como parámetro de entrada el nuevo enlace l_{ij}.

A continuación se explican en detalle cada uno de los distintos métodos mencionados en los casos expuestos con anterioridad y de los cuales se pueden encontrar las características principales en la Tabla 4.1:

- *link Status To False* (Figura 4.4): este método será el encargado de decidir si se regenerará el olor desde el enlace borrado (borrar y difundir de nuevo) o si únicamente se borrará el rastro. Para ello, tal y como se puede ver en la Figura 4.4, se elegirá el nodo con menor cantidad de olor del enlace bajo estudio (l_{ij}), es decir, *i* o *j*. Una vez elegido el nodo, si el olor del mismo es mayor que el umbral de la cantidad mínima de olor del tipo bajo estudio (u_i) se llamará al método *regenerate Odor Link* (Figura 4.5) para que borre el rastro de olor que parte de dicho nodo y regenerarlo siguiendo el procedimiento explicado para la difusión inicial del olor; y en caso contrario, siempre que el olor sea mayor que cero, se llamará a *delete Odor* (Figura 4.8) para borrar el rastro de olor que se encuentra en los nodos que dependen del nodo elegido para tenerlo.

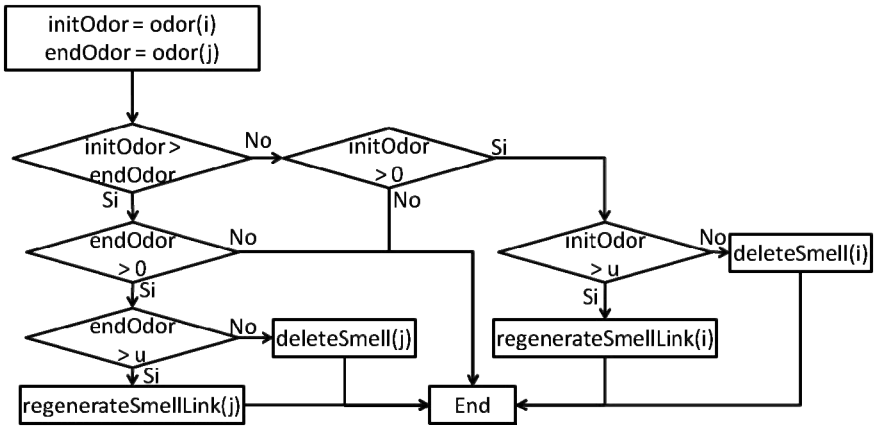


Figura 4.4. Diagrama de flujo de *link Status To False*.

- *regenerate Odor Link* (Figura 4.5): este método será el encargado de borrar el rastro de olor que parte del nodo con menos olor del enlace borrado (*i* en la Figura 4.5) y de llamar al método *diffusion* (Figura 4.9) para difundir de nuevo el olor desde dicho nodo.

Con el objetivo de realizar el borrado, se irán seleccionando nodo por nodo aquellos que van uniéndose con *i*, empezando con los nodos adyacentes y terminando con aquellos con menor olor que también tienen olor gracias al nodo *i*. Para ello, se comprobará si cada uno de los nodos adyacentes del nodo bajo estudio (que en la Figura 4.5 toma el valor de *nodeID*) tiene un olor inferior o igual que el olor de dicho nodo (*aux* en la Figura 4.5) menos el coste del enlace que los une, y en caso de cumplirse, se eliminará el olor del nodo adyacente que se está estudiando (*n(index)* en la Figura 4.5). Una vez que se ha puesto a cero el olor en todos los nodos que tienen olor gracias a *i*, se iniciará una difusión de olor desde dicho nodo (*diffusion(i)* en la Figura 4.5).

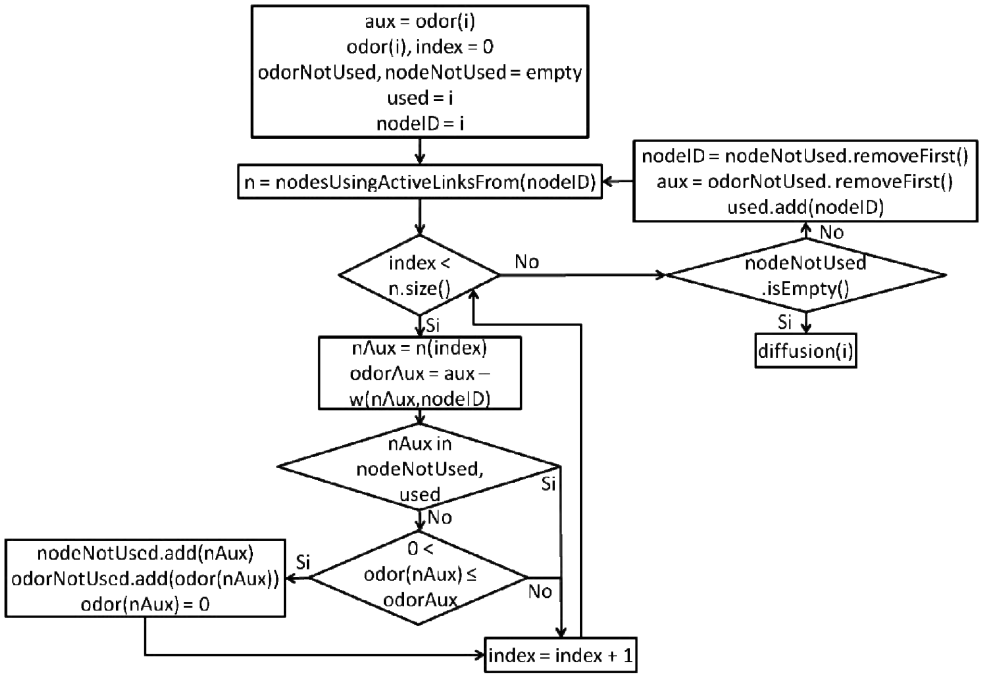


Figura 4.5. Diagrama de flujo de *regenerate Odor Link*.

- *link Status To True* (Figura 4.6): la llamada a este método se realiza cuando se crea un nuevo enlace (l_{ij}) entre los nodos *i* y *j*. En este caso, lo que se hará será comprobar si el olor de alguno de los nodos extremos del nuevo enlace creado es mayor que el valor de *u* del tipo de olor bajo estudio (u_i). En caso afirmativo, se llamara al método *diffusion* (Figura 4.9) pasándole como parámetro el nodo elegido para que comience la difusión de olor desde él (*diffusion(i)* o *diffusion(j)* en la Figura 4.6).

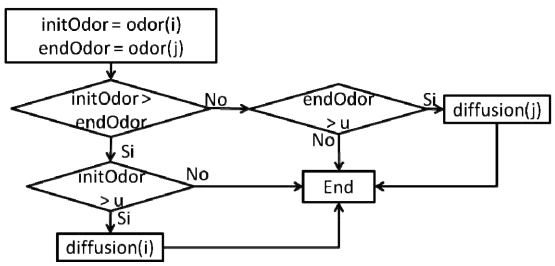


Figura 4.6. Diagrama de flujo de *link Status To True*.

- *regenerate Odor Node* (Figura 4.7): cuando se borra un nodo del grafo (*i* en la Figura 4.7) que tiene para algún tipo de olor del grafo una cantidad del mismo mayor que el u_i correspondiente, se llamará a este método para que el olor de todos los nodos que adquirieron alguna cantidad del mismo gracias a él pasen a tener un olor igual a cero. El procedimiento a seguir para realizar este borrado es el mismo que se explicaba en *regenerate Odor Link*, pero con la diferencia de que en este caso se almacenarán los nodos adyacentes a él y que tengan un olor menor o igual que el suyo menos el coste del enlace que los une (en la Figura 4.7 *aux – w(nAux, nodeID)*). Esto es así, porque será desde estos nodos (almacenados en *bigger Odor* en la Figura 4.7) desde los que se hará la difusión del olor (*diffusion (bigger Odor)* en la Figura 4.7).

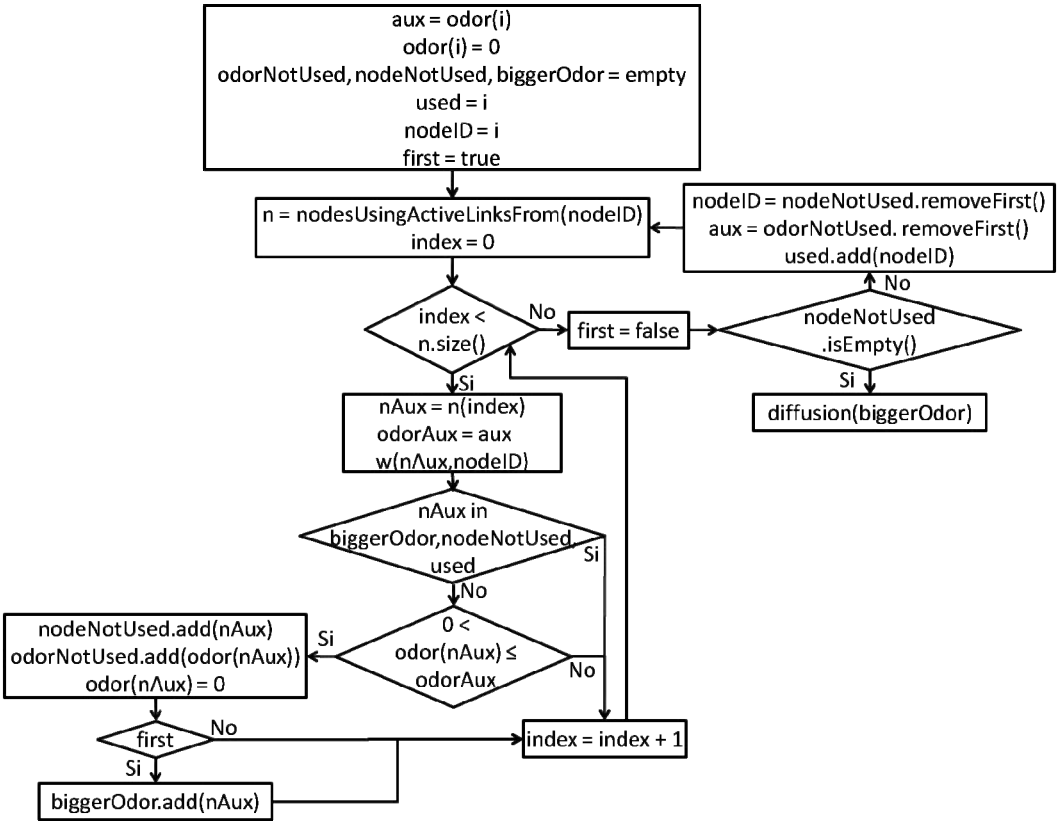


Figura 4.7. Diagrama de flujo de *regenerate Odor Node.*

- *delete Odor* (Figura 4.8): a este método se le llamará tanto en el caso del borrado de un enlace (pasando como parámetro de entrada el nodo elegido en *link Status To False*) como de un nodo cuando la cantidad de olor asociada sea menor que *u* y mayor que cero. El procedimiento a realizar es el mismo que el explicado en *regenerate Odor Link*, pero en este caso no se llamará a *diffusion*, pues no se tendrá que reiniciar la difusión del olor (el nodo desde el que se realizará el borrado no formaba parte del plan de difusión de olor inicial que comprendía nodos con olor únicamente entre *m* y *u*).

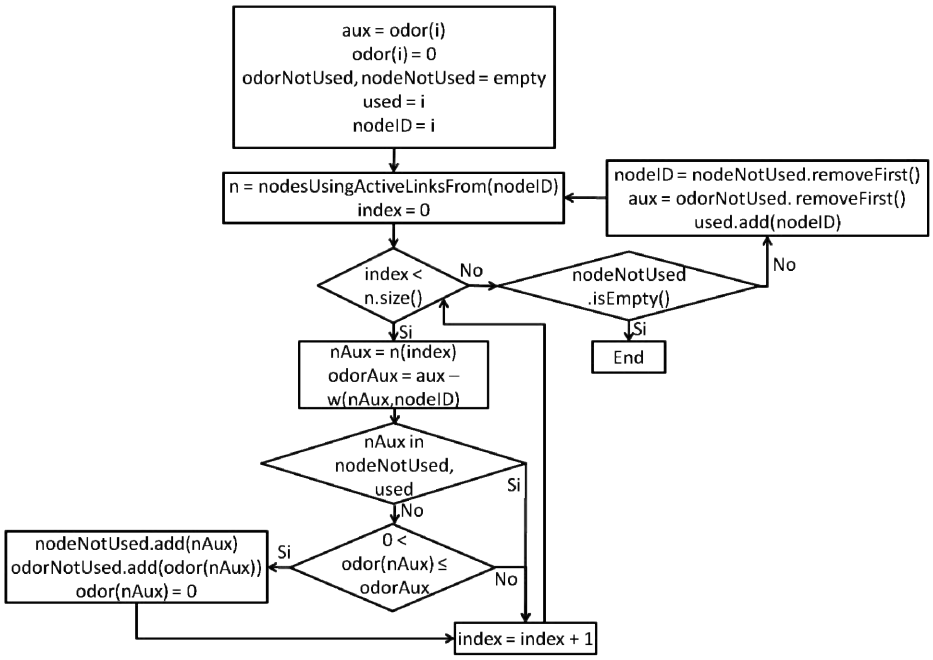


Figura 4.8. Diagrama de flujo de *delete Odor*.

- *diffusion* (Figura 4.9): este método será el encargado de realizar una difusión de un olor de un tipo determinado desde la lista de nodos introducida como parámetro (*bigger Odor* en Figura 4.9). La forma de realizar esta difusión se basará en los principios de la difusión de olor que se explicó en el apartado 3.3. Para ello, en cada iteración, se elegirá el nodo con mayor cantidad de olor que no haya sido utilizado hasta ese momento para difundir olor desde él, y se dispersará olor a sus nodos adyacentes siempre y cuando el olor a asignar sea mayor que el u_i correspondiente y menor que el olor que tuviese asignado ese nodo adyacente.

  Cuando ya no sea posible cumplir la condición de que el olor a asignar a ningún nodo sea mayor o igual que u_i, la difusión terminará.

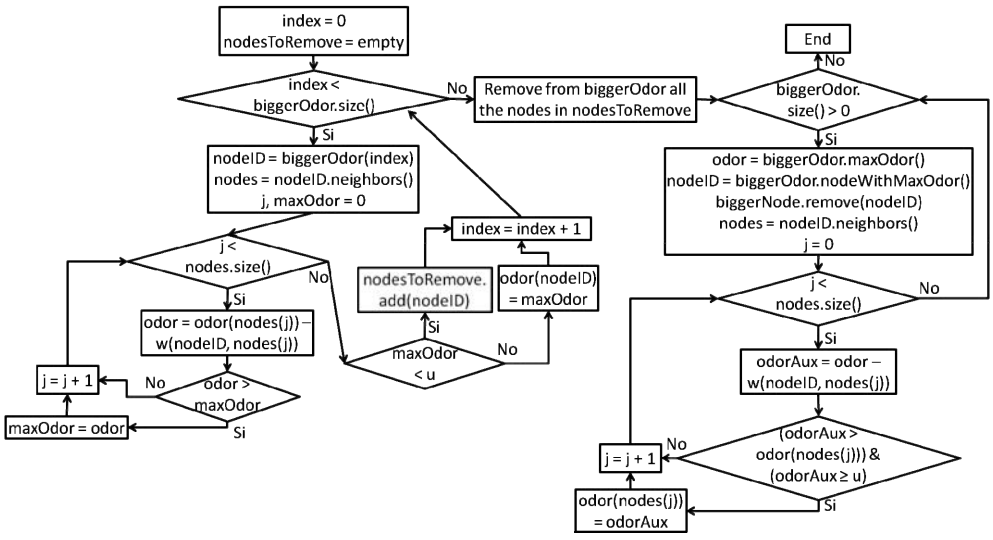


Figura 4.9. Diagrama de flujo de *diffusion*.

Una vez explicados todos los métodos que forman parte de la modificación de los nodos y enlaces cuando los mismo afectan al olor, se pasa a mostrar de manera gráfica un ejemplo del caso que se cree más completo: borrado de un nodo con olor mayor que u_i (en este caso $u_i = 16$) y menor que m ($m = 20$). Dicho ejemplo se muestra en la Figura 4.10, y se pasa a explicar a continuación.

Inicialmente se parte de un grafo que se encuentra en el estado mostrado en la Figura 4.10 (a). En ella se puede observar que además del área que comprendería la difusión de olor inicial (todos los nodos con olor mayor o igual que u_i) existe una dispersión de olor radial de manera que hay nodos con olor que va de 16 a 12.

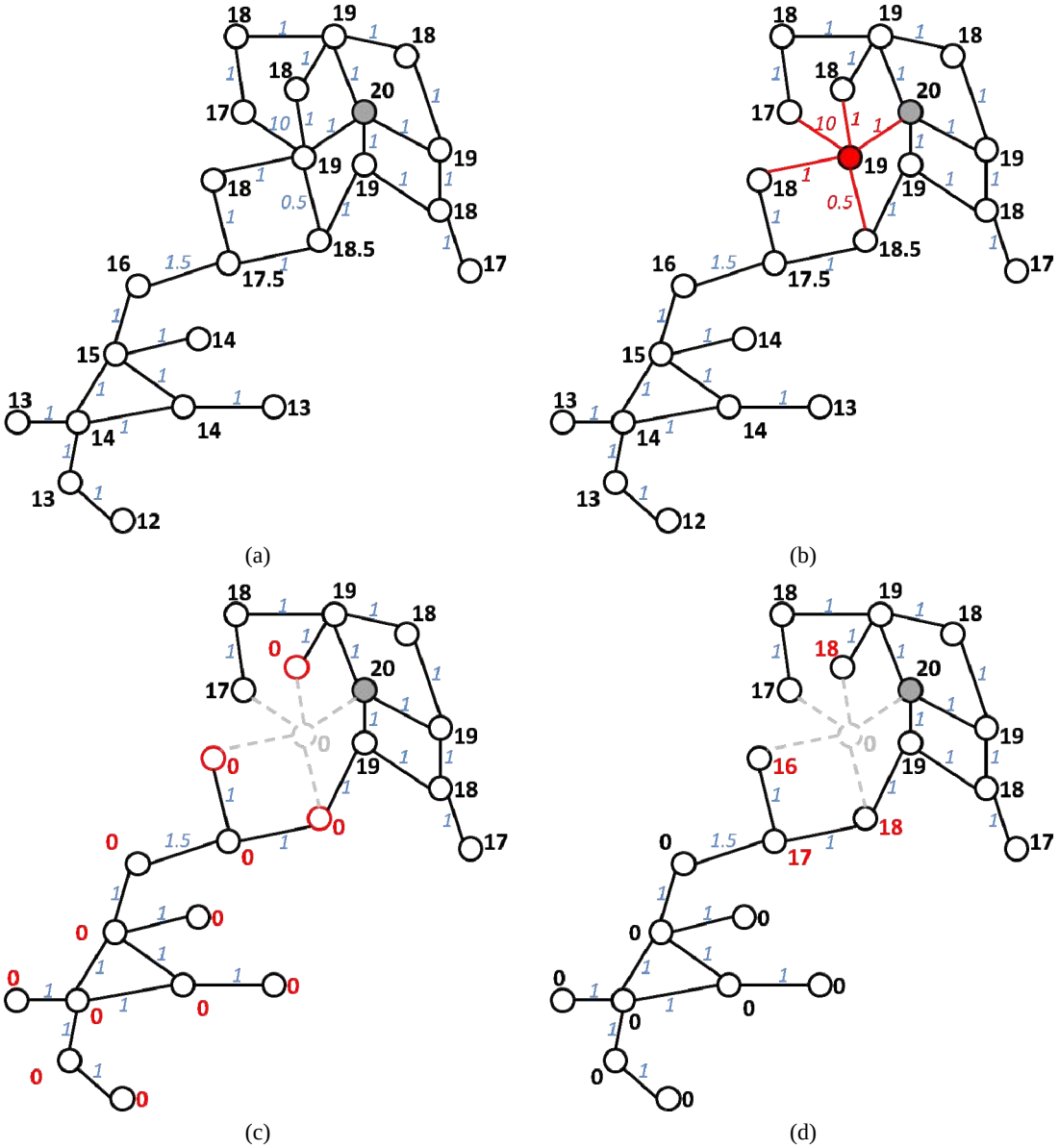


Figura 4.10. Ejemplo de regeneración de olor al eliminarse un nodo con olor en el rango *[m, u]*.

En un determinado momento, el nodo marcado en rojo en la Figura 4.10 (b) se elimina del grafo. Puesto que es la eliminación de un nodo, también se eliminarán los enlaces que lo tienen

como uno de sus nodos extremos (enlaces marcados en rojo en Figura 4.10 (b)). Una vez hecho esto, se llamaría a *regenerate Odor Node*, pues el olor del nodo es 19 que es menor que *m* pero mayor que *u*, pasándole como parámetro de entrada el identificador del nodo eliminado.

Después de la ejecución de este método, el grafo se encontrará en el estado mostrado en la Figura 4.10 (c), en la cual todos los nodos que tenían olor menor que el del nodo borrado, y cuya cantidad de olor dependía de él, aparecen con olor igual a cero. Es decir, todo el olor radial desaparece, así como el olor de algunos nodos que se encontraban en el área de olor inicial. Además de esto, se seleccionarán los nodos adyacentes al nodo eliminado y cuyo olor era menor que el suyo y dependientes de él (nodos marcados mediante un contorno rojo en Figura 4.10 (c)). Esta selección es debida a que será desde dichos nodos desde los que se llevará a cabo la difusión de olor realizada por el método *diffusion*, que tratará de volver a asignar olor a aquellos nodos que lo perdieron y que formaban parte del área con olor inicial.

El resultado de esta difusión de olor se muestra en la Figura 4.10 (d). En la misma se puede apreciar como aparecen cuatro nodos a los cuales se les ha reasignado una nueva cantidad de olor mientras que 9 nodos que anteriormente tenían olor, dejan de tenerlo debido a que la difusión solo alcanza a aquellos nodos que tuviesen un olor mayor o igual que u_i.

4.3 Versión Final de la Propuesta y Parámetros Principales

Una vez terminados los ciclos de evolución de la propuesta para conseguir los objeticos planteados inicialmente; se puede resumir concluyendo que *SoSACO* es un algoritmo basado en el algoritmo ACO básico, aunque podría estar basado en cualquier otra versión de este algoritmo, al cual se incluye una ayuda en la búsqueda, para disminuir la tasa de pérdida de las hormigas al trabajar sobre grafos vastos así como para disminuir el tiempo de respuesta, que actúa a modo de puntos de encuentro, *Nodos Comida*, fácilmente localizables en el grafo gracias a una cualidad numérica atribuida a los nodos del grafo, *Olor a Comida*, que actúa a modo de rastro a seguir y cuyo seguimiento en orden creciente indica cómo alcanzar el *Nodo Comida*.

Además de esto, puesto que las hormigas se dividirán en subcolonias que partirán de nodos extremo distintos, una primera aproximación al camino solución se podrá obtener cuando dos hormigas, una de cada colonia, alcancen el mismo *Nodo Comida*, o distinto, ya que el camino global se podrá obtener mediante la concatenación de los caminos parciales encontrados y los que se preprocesan entre los nodos comida.

Por último, con el fin de controlar el dinamismo propio de los grafos sobre los que se trabajará, se incorporan una serie de procedimientos para actualizar el olor asignado a cada nodo así como para evitar que un número elevado de hormigas *zombi* afecte de manera negativa a la resolución de un servicio gracias a la creación de unas hormigas nuevas que las sustituyen

(aunque las hormigas *zombi* sigan buscando por no ser viable el identificarlas en cada momento).

Con el fin de que *SoSACO* funcione de una manera lo más correcta posible, el analista tendrá que dar valor a una serie de parámetros relevantes en la forma de funcionar del algoritmo:

- $F = \{f_i\}$: Conjunto de Nodos Comida en el grafo, es decir, aquellos nodos a los que, por cumplir unas determinadas características, será útil llegar de una manera rápida. Por ejemplo, nodos con un alto grado o frecuentemente empleados como nodo destino.
- $|F|=|S|$: Número de Nodos comida en el grafo, que es igual al número de tipos de olor, y por tanto al número de áreas con olor que existen en el grafo.
- $U = \{u_i\}$: Valor mínimo de cada tipo de olor asignable a un nodo cuando se realiza la dispersión inicial de olor, y que por tanto determinará el tamaño de cada s_i.
- k: Peso que tienen los costes de los enlaces en la reducción del olor cuando se pasa de un nodo a otro de un enlace.
- m: Valor máximo de olor que existe en el grafo, y que por tanto se encontrará en los nodos comida (f_i) por ser desde estos desde los que se dispersa el olor.
- h: Factor que pondera la feromona asociada a un elemento borrado del grafo para determinar el número de nuevas hormigas que debe crearse.

Capítulo 5 Evaluación

Una vez explicado el Planteamiento (primera fase del ciclo) y el Análisis y Diseño (segunda fase del ciclo) de todos los ciclos que van a formar parte de la evolución del algoritmo *SoSACO*, se va a proceder a explicar la Evaluación (tercera fase del ciclo) que se realizó en cada uno de ellos junto con las Conclusiones (cuarta fase del ciclo) extraídas de cada uno de los experimentos. Puesto que cada evaluación se corresponde con un ciclo, el orden en el que se van a mostrar los resultados será el mismo que el seguido en el Capítulo 4.

Precediendo a la evaluación de cada uno de los ciclos, se incluirá una descripción de los objetivos que se persiguen con la misma (apartado 5.1) así como una explicación del escenario genérico de pruebas en el apartado 5.2. Además, se incorporará un apartado en el que se discutirá de manera conjunta todos los resultados obtenidos (5.4) y uno en el que se ejecutará el algoritmo resultante sobre un escenario de pruebas real (5.5), en concreto una red social.

5.1 Objetivos

Con la evaluación llevada a cabo en cada uno de los ciclos se pretende comprobar si los objetivos enumerados en el capítulo de Introducción se han alcanzado en la propuesta. Es decir, quiere demostrarse que se ha conseguido un algoritmo capaz de dar una solución empleando el menor tiempo de respuesta posible cuando las búsquedas de caminos se realizan sobre grafos vastos dinámicos, así como demostrar que el algoritmo puede funcionar en grafos de topología genérica.

Con respecto a los otros dos objetivos marcados en el capítulo de Introducción (manejo eficaz de la cantidad de datos con la que se trabaja e independencia entre la gestión de la información almacenada y el algoritmo de búsqueda), no se va a realizar ningún tipo de evaluación en este documento. Esto es así porque ambos objetivos quedan satisfechos con el refinamiento realizado a la base de datos que se lleva a cabo mediante la utilización del gestor que incorpora la misma. De esta manera, el problema de gestión del almacenamiento se aísla del

de la búsqueda de un algoritmo eficiente, focalizándose este trabajo de tesis doctoral en esto último.

Una vez dicho esto, con el fin de comprobar si se satisfacen los objetivos relativos al algoritmo de búsqueda, se analizará en cada uno de los ciclos de vida tres parámetros (en media, mediana, o desviación típica): *coste del camino*, *tiempo de respuesta*, y *tasa de éxito*, de manera que se asignará a los distintos parámetros con relevancia en la propuesta (los indicados en el apartado 4.3) el valor que permita que el algoritmo requiera unos tiempos de respuesta bajos con un coste del camino lo más cercano posible al óptimo y con unas tasas de éxito elevadas.

5.2 Diseño del Experimento

Puesto que en cada ciclo de evolución del algoritmo se plantearán una serie de alternativas a incluir en el mismo, será necesario incorporar a cada uno de estos ciclos una evaluación comparativa con el fin de seleccionar una de ellas, aquella que permita el mejor comportamiento del algoritmo a la hora de buscar caminos dentro de grafos. Para ello, cada uno de los ciclos contará con detalles concretos de evaluación, sin embargo, hay rasgos comunes en las pruebas que serán detallados en esta sección.

La primera característica común es el grafo sobre el que se realizarán las pruebas. Éste contará con 200.000 nodos y 600.000 enlaces. Con respecto a la topología, aunque inicialmente se pensó en utilizar una topología genérica, se optó por una que tratase de simular una *Small-World*, que por ser más específica incluía un abanico mayor de características. Por este motivo, se eligió una distribución en la cual existe un alto valor del coeficiente de agrupamiento en el centro del grafo aunque el número de pasos a dar para alcanzar el destino es mucho mayor que en dicho tipo de topologías. Es decir, se someterá al algoritmo a un entorno que no le es del todo favorable. En lo referente al número de nodos comida, así como del tamaño que alcanzarán sus áreas de olor, dependerá de la fase de pruebas.

Con respecto a los parámetros con influencia en el comportamiento del algoritmo, se tomarán los que se muestran en la Tabla 5.1:

- Un tiempo máximo de ejecución ($t_{threshold}$) de 700 seg. Esto es, el tiempo que esperará como máximo un usuario para obtener un primer resultado del camino. El hecho de elegir este valor se debe a que con mayor tiempo de ejecución se podrían emplear algoritmos como Dijkstra que obtienen el camino óptimo.
- 500 hormigas corriendo con el fin de encontrar un camino entre el nodo inicio y el destino. El elegir este número de hormigas y no uno menor se debe a que con menos hormigas la calidad del camino obtenido era peor y había una mayor tasa de pérdida de las hormigas, no pudiendo dar un resultado al servicio solicitado. Con respecto a la cota

superior, era el número máximo de hilos que permitía la máquina sobre la que se trabajó. Un estudio más detallado acerca de la elección del número de hormigas a emplear puede encontrarse en [Rivero et al., 2011b].

- En cuanto a la tasa de evaporación (ρ), se eligió un valor relativamente alto porque no se asociaron heurísticas a la fórmula para elegir el siguiente nodo al que moverse (Ecuación 4.4), de modo que con el fin de no caer en mínimos locales, la tasa de evaporación tenía que tomar un valor elevado.
- Con respecto al valor del máximo olor a comida que se va a depositar (m) se ha elegido el mostrado en la Tabla 5.1 por permitir una dispersión de olor que puede alcanzar a cualquier nodo del grafo (teniendo en cuenta que el peso que tendrá cada enlace recorrido en la difusión del olor, k, es del 100%).

Parámetro	Valor
$t_{threshold}$	700 seg
#ants	500
ρ	0,6
m	1.000.000
k	100%

Tabla 5.1. Parámetros del algoritmo.

Además de estos parámetros, habrá dos factores más que afectarán a la forma de funcionar del algoritmo. Estos tienen que ver con el momento en el que se reiniciará el olor, para eliminar el rastro del mismo depositado por los caminos entre los nodos inicio/destino que hayan utilizado algún nodo comida, y aquel en el que se reiniciarán los rastros de feromona.

Con el fin de que estos momentos queden claros, en primer lugar es necesario explicar cómo se van a organizar los servicios que se ejecutarán sobre el grafo. Para ello, se ha decidido que en cada fase de experimentación van a realizarse diez consultas con mil servicios cada una de ellas (número que se considera suficiente para aportar resultados representativos dado el tamaño del grafo) garantizando que dentro de una consulta no se repetirá ningún servicio. Cada una de las consultas se repetirá además cuatro veces para obtener resultados estables debido a que el algoritmo es estocástico.

Una vez indicado esto, los depósitos de olor se reiniciarán cada vez que se termine una consulta, no al finalizar cada servicio. Esto permitirá ver la evolución del efecto del incremento de los tamaños de las áreas de olor en la calidad del camino obtenido y en el tiempo de respuesta a lo largo de la consulta.

En lo referente a los rastros de feromona, serán reiniciados al comienzo de cada servicio. La explicación a la elección de este momento es que los servicios no siguen ninguna distribución, de modo que los nodos inicio/destino de uno no se encuentran cerca de los siguientes, pudiendo

llevar a una búsqueda errónea el guardar los rastros de feromona que se obtuvieron al finalizar el servicio anterior.

Para finalizar, habrá en experimentos en los que será necesario realizar una comparativa de los resultados obtenidos en este algoritmo con respecto a otros existentes. En estos casos, se mostrará una comparación de los resultados de *SoSACO* con los obtenidos por Dijkstra.

El elegir este algoritmo se debe a que es el clásico algoritmo de búsqueda de caminos (siendo el que normalmente proporcionan los Sistemas Gestores de Bases de Datos para realizar tal función), y porque el coste de los caminos que calcula es el óptimo, de modo que va a permitir saber cuánto difiere la solución obtenida con el algoritmo propuesto de este valor. Además de estos motivos, hay que añadir el que no se encontró ningún algoritmo que se adaptase a las características del problema que se está intentando resolver en este trabajo de tesis doctoral, por lo que se decidió emplear un algoritmo altamente conocido en la búsqueda de caminos como *benchmarking* y que además es empleado por varios de los trabajos presentados en el estado del arte.

Inicialmente también se pensó en realizar una comparativa con los resultados obtenidos con el algoritmo ACO, ya que es el algoritmo del que parte *SoSACO* y podría dar una idea de si se ha incluido realmente una mejora con respecto a él o no. No obstante, como se podrá apreciar en las posteriores fases de experimentación, estos resultados no se muestran. La razón es que dicho algoritmo tiene un comportamiento pésimo en el grafo de pruebas, arrojando una tasa de éxito por debajo del 30%. Además, en aquellos servicios en los que se obtenía un camino, la calidad del mismo era baja. El principal problema es que las hormigas se perdían durante la búsqueda debido a que el número de pasos a dar en el grafo no es bajo, y que en caso de hallar un camino, el mismo se encontraba de una manera aleatoria, por lo que su calidad difería del valor óptimo.

5.3 Ejecución y Resultados

Como se pudo ver en el apartado 3.5 (Metodología), cada ciclo del proceso de evolución de la propuesta va a tener una fase de Evaluación y Conclusiones. En este apartado se mostrarán los resultados obtenidos de la experimentación llevada a cabo en cada uno de los ciclos desarrollados así como una serie de conclusiones extraídas de estos.

5.3.1 Grafo Estático: Un Nodo Comida

Con el fin de comprobar cuál de las combinaciones de todos los elementos explicados en el apartado 4.1.1 es el que presenta un mejor comportamiento a la hora de buscar caminos dentro de grafos, se va a pasar a mostrar una serie de experimentos, ejecutados sobre el entorno de pruebas que se explicó en 5.2, en los que únicamente se contará con un nodo comida (f_1), situado en el nodo con mayor grado, y por lo tanto con un único área de olor (s_1).

Lo primero que se trató de fijar fue la utilización que se iba a hacer de la *lista tabú*. Tal y como se explicó en 4.1.1.3, se iban a tener en cuenta dos tipos de control de la *lista tabú*: no utilizarla, o aplicar el control *β-tabú*. En concreto, se va a realizar una comparación de los resultados obtenidos cuando no se utiliza la *lista tabú*, cuando se tiene *1-tabú* (en cada paso la hormiga intentará moverse a un nodo que no esté en la *lista tabú*), y cuando se tiene *2-tabú* (solo se intentará evitar utilizar los nodos visitados con anterioridad en los pasos múltiplos de dos). Con respecto a la *lista tabú* estricta no se tendrá en cuenta pues, tal y como se explicó en el apartado 4.1.1.3, la tasa de éxito en la búsqueda del camino era baja.

Para ello, a los parámetros explicados en 5.2, se añadirá que el valor mínimo de olor elegido será aquel que ponga a las hormigas en la peor situación posible, aquella en la que tienen que recorrer más enlaces sin ayuda para alcanzar el destino, con el fin de probar correctamente el mejor tipo de control de la *lista tabú*. Por este motivo se eligió *u = 999.700*, que implica un área de olor que cubre al 1% de los nodos del grafo.

Teniendo en cuenta todo esto, los resultados en coste y tiempo del camino obtenido después de ejecutar todas las consultas son los de la Tabla 5.2.

Observando los resultados del coste, Tabla 5.2 (a), se puede apreciar que los valores de la media y la mediana son prácticamente iguales, observándose que es algo mejor el caso de la *2-tabú*. No obstante, teniendo en cuenta la desviación típica, los resultados en coste pueden considerarse los mismos en todas las versiones.

Con el fin de decidirse por una de las versiones, se pasará a estudiar el tiempo de respuesta, Tabla 5.2 (b). En lo referente a este parámetro, lo más llamativo es la desviación típica, pues en todos los casos tiene un valor elevado. Esto refleja un buen comportamiento medio aunque implica una pérdida eventual de rendimiento en muchos casos, por lo que se recomienda la solución basada en *1-tabú*, que presenta un rendimiento más equilibrado que el resto.

	Sin *Lista Tabú*	***1-Tabú***	***2-Tabú***
Mean	9413,10	10899,53	8248,87
SD	1415,43	2223,58	1277,33
%SD	15,04%	20,40%	15,48%
Median	9334,05	10708,16	8246,26

(a) Coste

	Sin *Lista Tabú*	***1-Tabú***	***2-Tabú***
Mean	3353,01	2413,30	7919,05
SD	7630,43	3400,91	19070,05
%SD	227,57%	140,92%	240,815%
Median	2009,50	1966,58	2738,8

(b) Tiempo (mseg)

Tabla 5.2. Datos estadísticos del tiempo y coste de los caminos encontrados con distintos controles de la *lista tabú*.

A nivel de la tasa de éxito, todos los tipos de control de *lista tabú* empleados presentan un 0% de servicios sin resolver (la única con un valor distinto del 100% es la *2-tabú*, con una tasa de éxito del 99,6%, es decir, que puede considerarse del 100%).

Por todos estos motivos, se concluyó que lo mejor era utilizar la ***1-tabú***, pasando a estudiar el comportamiento del algoritmo en función de los distintos valores de *u* para fijar uno y de esta

forma poder elegir, posteriormente, el tipo de actualización de feromona así como el tipo de división de las colonias.

Para ello, se plantearon tres posibles valores del límite inferior del olor: $u = 999.700$ (porcentaje de nodos en s_1 del 1%), $u = 998.600$ (porcentaje de nodos en s_1 del 17%), $u = 998.000$ (porcentaje de nodos en s_1 del 30%).

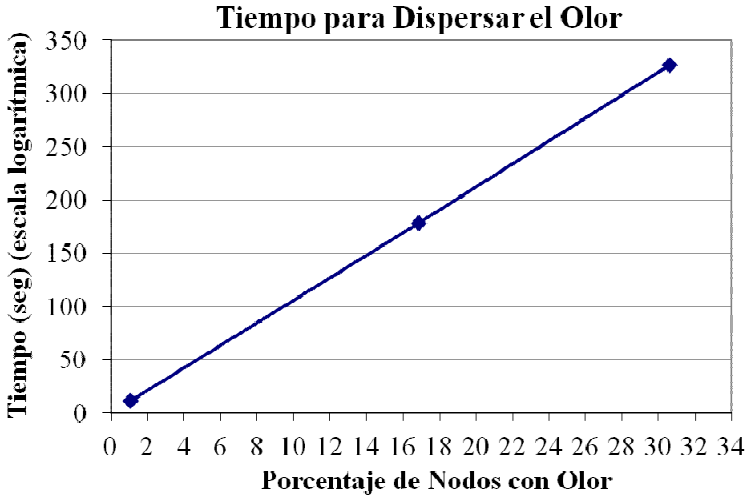


Figura 5.1. Tiempo de dispersión de olor.

Puesto que el algoritmo tiene que adaptarse en posteriores versiones a un grafo dinámico, lo primero que es importante observar es el tiempo que es necesario emplear en la creación de s_1.

Este tiempo aparece en la Figura 5.1. Por cada uno de los distintos u se van a obtener unos valores de $|s_1|$, de manera que cuanto menor sea u, mayor será el número de nodos que contenga. Esto va a provocar un aumento del tiempo requerido para realizar la dispersión, pues se tienen que alcanzar más nodos.

Una vez visto esto, se van a realizar las consultas de búsquedas de caminos con el fin de decidir si el tiempo extra que requieren en la dispersión de olor los valores de u menores implica mejoras suficientes como para emplearlos.

En este caso, por servicio se va a entender una petición de búsqueda de camino entre dos nodos del grafo donde el nodo destino será f_1 y el nodo inicio será elegido de manera aleatoria de entre todos los existentes en el grafo, garantizando que dentro de una consulta no se repita ninguno. El haber decidido que todos los nodos destino sean f_1 pretende simular el hecho de que en la vida real se suele ir con más frecuencia a los nodos de interés de la aplicación que al resto, de modo que es importante ver cómo funciona el algoritmo propuesto en estos casos. Además, al darse este caso de nodo destino, la colonia no se debe dividir, y por lo tanto no influye el tipo de división en la solución a elegir en este caso.

Resaltar también que sólo se va a tener en cuenta el primer camino obtenido por las hormigas. Es decir, una vez encontrada la primera solución, el servicio se dará por finalizado y se pasará al siguiente.

Por último, los valores que van a tomar los principales parámetros del algoritmo, serán los mismos que se empleaban para decidir entre los distintos tipos de control de la *lista tabú* salvo el valor de *u* que se indicaba (pues es el parámetro que varía en este caso). El control de la *lista tabú* empleada será la que se decidió en el estudio anterior (*1-tabú*). Además, en este caso, se va a mostrar una comparación de los resultados obtenidos con *SoSACO* frente a los de Dijkstra. Una vez dicho esto, se pasan a estudiar los resultados obtenidos.

En la Figura 5.2 se muestra como, tanto el tiempo de respuesta, como el coste del camino obtenido disminuye según van ejecutándose los servicios. Esta disminución es importante sobre todo en los 50 primeros servicios, en los que se observa una pendiente de bajada pronunciada, ya que después existe una estabilización. Esto es importante, ya que pone de manifiesto la relevancia de la dispersión radial de olor realizada al final de cada servicio.

Con respecto a la influencia de los valores de *u* en dicha evolución, se puede concluir que no es significativa: para todos los valores de *u* el algoritmo se estabiliza, aproximadamente, a partir del mismo servicio. La única diferencia observable es la del valor del coste al que se estabilizan, que es ligeramente menor cuanto menor es *u*. No obstante, esto se estudiará en mayor detalle a continuación.

En cuanto al tiempo de respuesta, se puede observar cómo, en cualquiera de los servicios, incluso en los primeros, siempre es mucho menor al requerido por Dijkstra. Esto se podría ver enturbiado por el coste del camino encontrado. Este defecto se va a estudiar más en detalle en las siguientes gráficas y tablas para analizar su relevancia.

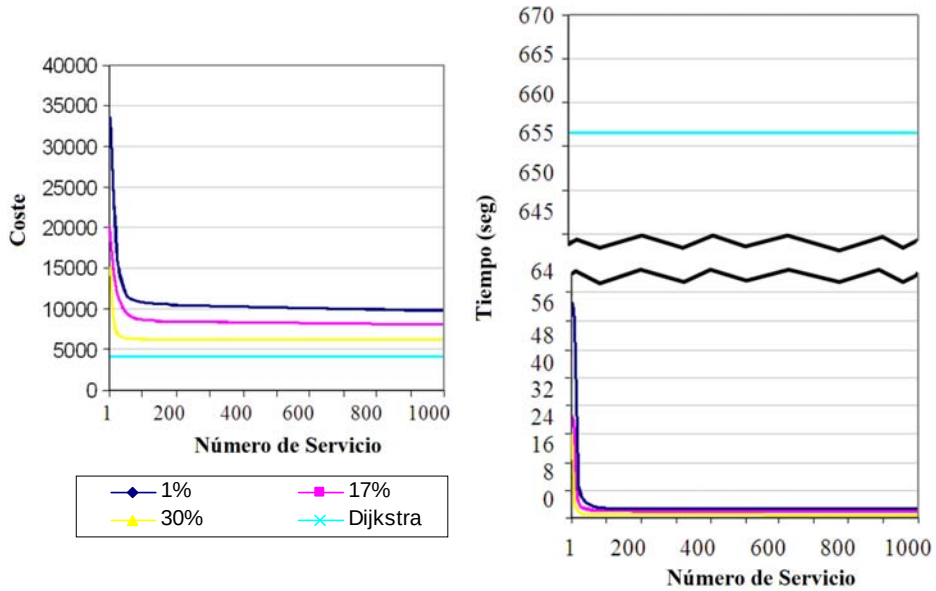


Figura 5.2. Coste y tiempo para obtener el primer camino en cada servicio (líneas de tendencia).

Por ejemplo, en la Tabla 5.3 se muestran una serie de datos estadísticos que permiten ver con más claridad las diferencias existentes entre Dijkstra y el nuevo algoritmo, así como entre los distintos valores de *u* empleados en *SoSACO*. En dichas tablas se puede apreciar como las medias de los costes obtenidos con el nuevo algoritmo con sus distintos valores de *u* son superiores al óptimo, no obstante, dicha diferencia (que es menor según aumenta el número de nodos con olor) se ve recompensada con la enorme diferencia existente en tiempo. Una solución a este problema podría ser el dejar ejecutar durante un tiempo mayor a las hormigas, ya que como se indicó anteriormente sólo se ejecutan hasta encontrar un primer camino, aunque en futuras evoluciones del algoritmo se intentará solventar este problema.

En lo referente a la desviación típica en cada uno de los algoritmos, se puede apreciar como todos ellos tienen un valor relativamente bajo, siendo el de Dijkstra el mejor de todos.

	Dijkstra	**1%**	**17%**	**30%**
Mean	3047,16	10899,53	7676,30	6015,43
SD	520,40	2223,58	1817,56	1520,62
%SD	17,08%	20,40%	23,68%	25,28%
Median	3016,14	10708,16	7591,04	5895,64

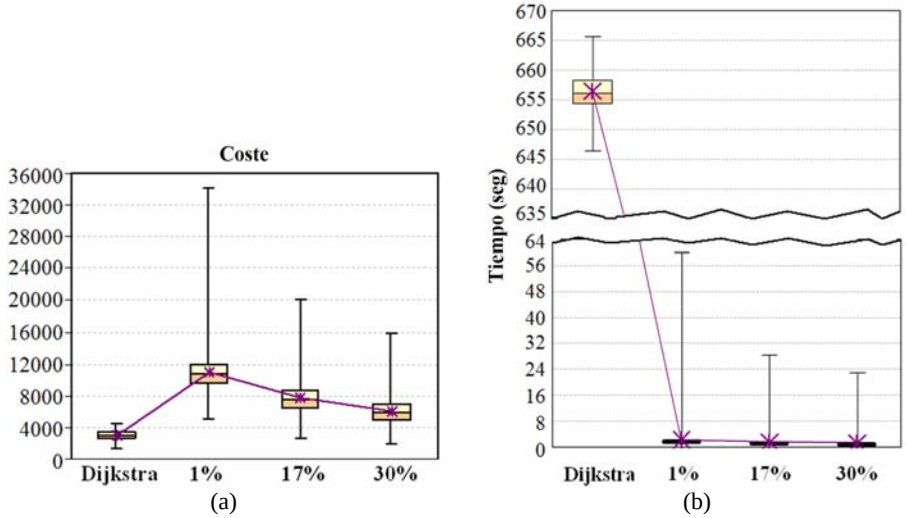
(a) Coste

	Dijkstra	**1%**	**17%**	**30%**
Mean	656461,53	2413,30	1710,90	1375,55
SD	2917,98	3400,91	1857,64	1465,53
%SD	0,44%	140,92%	108,58%	106,54%
Median	656230,90	1966,58	1413,58	1135,68

(b) Tiempo (mseg)

Tabla 5.3. Datos estadísticos del tiempo y coste de los caminos encontrados con distintos valores de *u*.

Con respecto a las diferencias apreciables en función del número de nodos con olor que existen en el grafo, cabe resaltar que, como se ve claramente en la Figura 5.3, la mejora introducida por el hecho de tener un mayor número de nodos con olor no es significativa, y si se tiene en cuenta la diferencia de tiempo existente entre expandir olor inicialmente en un 1% de los nodos o en un 30% de los nodos (Figura 3.4) dicha mejora es despreciable. Es decir, teniendo en cuenta que en el futuro ciclo de grafo dinámico, cada vez que hubiese un cambio que afectase a un olor, habría que regenerarlo, y hasta que no se finalizase este proceso las hormigas dispondrían de menos ayuda en la búsqueda, los beneficios obtenidos en coste/tiempo no se pueden considerar relevantes.



(a) (b)

Figura 5.3. Boxplot de tiempo de respuesta y coste para distintos valores de *u*.

Por último, como muestran la Tabla 5.3 o la Figura 5.3, el máximo valor representa un punto a eliminar de los datos, ya que es un valor anómalo. Por ello, se optó por realizar todos los cálculos estadísticos explicados anteriormente eliminando de los mismos el primer experimento para obtener unas conclusiones más claras. Dichos cálculos se pueden ver en la Tabla 5.4, en la que, como era de esperar, al eliminar el primer servicio, se ve una reducción tanto de la media del coste como del tiempo, así como en la desviación típica.

	Dijkstra	**1%**	**17%**	**30%**
Mean	3046,72	10712,67	7552,39	5942,99
SD	520,24	1700,41	1615,44	1422,78
%SD	17,08%	15,87%	21,39%	23,94%
Median	3016,11	10667,96	7543,55	5862,50

(a) Coste

	Dijkstra	**1%**	**17%**	**30%**
Mean	656461,08	2036,96	1474,59	1199,28
SD	2906,93	569,83	488,21	480,22
%SD	0,44%	27,97%	33,11%	40,04%
Median	656230,60	1948,40	1392,70	1122,35

(b) Tiempo (mseg)

Tabla 5.4. Datos estadísticos del tiempo de respuesta y coste sin primer servicio con distintos valores de *u*.

En cuanto a los tiempos, como se puede apreciar en la Figura 5.4, estos mejoran con respecto a Dijkstra, y ahora es más apreciable que prácticamente todos los resultados a los servicios solicitados se encuentran, en coste, próximos a los valores que se obtendrían con Dijkstra. No obstante, la diferencia sigue siendo alta debido a que solo se coge el primer camino obtenido, así como a que el número de enlaces que debe recorrer cada hormiga para alcanzar el destino es elevado por la estructura relativamente genérica del grafo de pruebas.

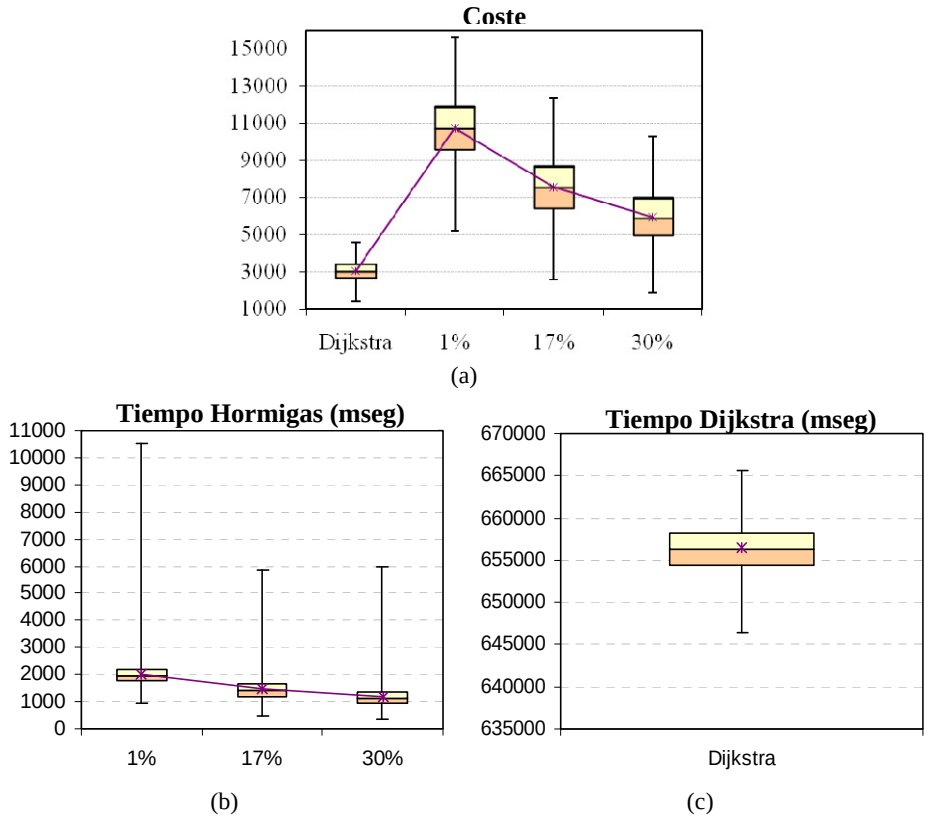
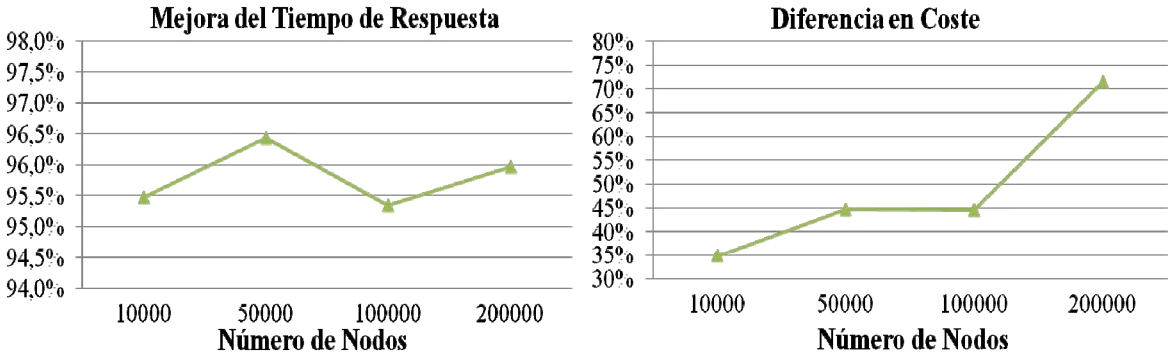


Figura 5.4. Boxplot de tiempo y coste sin primer servicio para distintos valores de *u*.

De acuerdo a los resultados presentados, *SoSACO* es competitivo cuando se implementa con un *u* alto, o lo que es lo mismo, un tamaño de s_1 pequeño, y con un *1-tabú*. Además, el hecho de

que sea mejor utilizar áreas de olor de tamaño pequeño va a permitir poder tener más de un nodo comida distribuido por todo el grafo pudiendo encontrar caminos más fácilmente, pues aumentará la probabilidad de tener un nodo comida cerca. Esto, además, solventará el problema que se explicó anteriormente de tener que recorrer grandes cantidades de enlaces para encontrar una solución.

Para completar el análisis, indicar que este algoritmo es escalable tal y como se puede observar en la Figura 5.5. En dicha figura se muestra la mejora en tiempo que proporciona el algoritmo propuesto con respecto a Dijkstra, Figura 5.5 (a), y la diferencia en coste, Figura 5.5 (b), según varía el numero de nodos que posee el grafo (el tipo de servicio, actualización, así como estructura del grafo es el mismo que en la experimentación anterior, y el número de enlaces será el número de nodos por tres). Lo que se muestra es que el tiempo de respuesta del algoritmo es prácticamente el mismo independientemente del número de nodos, y que el coste crece de manera lineal a medida que lo hace $|N|$ (aunque siempre con un resultado peor que el de Dijkstra). A esto hay que añadirle el hecho de que todos los servicios ejecutados en cualquiera de los grafos encontraron solución.



(a) (b)

Figura 5.5. Diferencia porcentual de *SoSACO* con respecto a Dijkstra al aumentar el número de nodos del grafo.

A continuación es necesario definir el tipo de actualización de feromona (local o global, así como la forma de actuar, en lo que a actualización de feromona se refiere, cuando el camino se completa utilizando el olor), la división de la colonia que se empleará, y si la colonia parará o no cuando se alcance un olor.

Persiguiendo elegir la mejor opción, el tipo de servicio va a cambiar con respecto al que se empleaba en la elección del valor del umbral mínimo de olor. En este caso, tanto el nodo destino como el inicio serán elegidos aleatoriamente dentro de todo el conjunto N eliminando del mismo a f_1. Esto se hará así para determinar si, por ejemplo, es bueno o no parar cuando se encuentra un olor, pues si el destino es el olor, siempre será mejor reiniciar la búsqueda, y la experimentación no aportaría ningún tipo de información para poder elegir.

Además del cambio en lo referente al tipo de servicio que se ejecuta, se incorporará otra variación más en la experimentación con respecto a la anterior, y es que se permitirá al

algoritmo ejecutar durante un tiempo extra después de producirse la primera solución (cuando existe una hormiga que ha encontrado un camino global, se permitirá que todas sigan ejecutando un tiempo fijo extra) con el fin de ver si los resultados mejoran. No obstante, el tiempo que se dejará de más a las hormigas estará limitado debido a que se necesitan respuestas rápidas a los servicios solicitados.

Con el fin de elegir un valor adecuado, se dejó ejecutar a las hormigas durante dos minutos y se estudió el tiempo en el cual obtenían una solución que no se mejoraba a lo largo del resto de ese tiempo limitado. Al realizar este estudio se obtuvieron los resultados recogidos en la Tabla 5.5, a los cuales es importante añadir que el 82,9% de los servicios se veían mejorados en coste. En ésta se indica el tiempo extra que requirió cada servicio para mostrar la mejor solución posible en los dos minutos que se le dejaba de margen, y la mejora en coste que se obtuvo. Puesto que se obtenía cierta mejora se decidió que era interesante dejar un tiempo de espera. Inicialmente se optó por la mediana, no obstante, como difería del valor de la media, se decidió emplear un valor ligeramente superior e igual a 15 segundos.

	Tiempo (mseg)	**Coste**
Mean	23.164,91	443,68
SD	24.335,96	930,73
%SD	105,06%	209,78%
Median	11.723,00	132,35

Tabla 5.5. Estudio del tiempo extra permitido.

Una vez dicho esto, se pasó a comprobar cuál era el mejor tipo de actualización a emplear: actualización local + global, o actualización global únicamente.

Para ello, los servicios empleados eligieron como nodos extremo aquellos que se encontraban alejados del s_1 pero cercanos entre sí con el fin de que toda la búsqueda dependiera de la feromona y no existiese ayuda por parte del olor para encontrar el camino (de manera que los resultados fuesen independientes de las versiones en las cuales las hormigas deben reiniciar su búsqueda o no cuando encuentran olor, o de la actualización de la feromona o no cuando el camino global utiliza s_1). Además, el hecho de no poder utilizar el olor, ponía al algoritmo bajo el peor escenario posible.

Los resultados de estos experimentos son los que se muestran en la Figura 5.6 (primer camino obtenido) y en la Figura 5.7 (mejor camino obtenido durante el periodo de los 15 segundos extra), en las cuales se desglosan los resultados en función del tipo de división de la colonia de hormigas (pues esto sí que puede influir en los resultados obtenidos). Es importante indicar que el eje vertical se ha representado en escala logarítmica para apreciar mejor los resultados que arrojan los experimentos, y que en el eje horizontal *GDivVen* representa al algoritmo *SoSACO* con Divide y Vencerás con Subdivisión y solo actualización global de feromona, mientras que *LyGDivVen* aplica actualización local y global; y *GSimple* es el

algoritmo con Divide y Vencerás Simple y solo actualización global, mientras que *LyGSimple* aplica actualización local y global.

Analizando los resultados se puede concluir que, independientemente del tipo de división de la colonia empleado, existe una mejora en el tiempo de respuesta y en el coste cuando el tipo de actualización de feromona incluye la actualización local y la global. Además, aunque la que solo incluye global mejora con el paso del tiempo, también lo hace la local + global, de modo que la mejor elección en cualquiera de los casos será la que actualiza la feromona paso a paso y cuando encuentra el camino entre los nodos extremo.

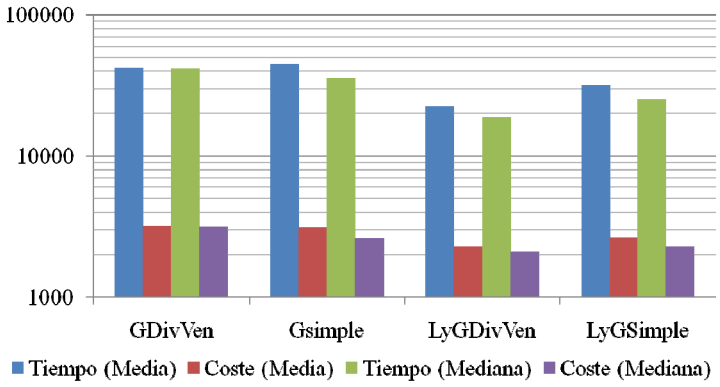


Figura 5.6. Primer resultado de coste y tiempo (mseg) para actualización global y local.

En cuanto al tipo de división de colonias a emplear, de los resultados obtenidos cuando no existe posibilidad de utilizar el olor, parece mejor el Divide y Vencerás con Subdivisión (los costes son menores tanto en el primer resultado como en el último y la mejor solución requiere menos tiempo de espera para encontrarse), no obstante, se esperará a ver los resultados obtenidos cuando los nodos inicio y destino están distribuidos por todo el grafo independientemente de que estén cerca o lejos de s_1, con la única restricción de que ninguno de ellos sea f_1, para decidir entre un tipo de división de hormigas u otro, pues será en estos casos en los que realmente se note un efecto de este parámetro en la forma de funcionar del algoritmo.

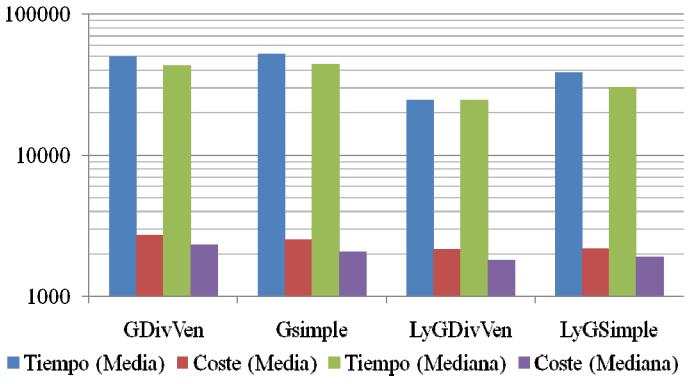


Figura 5.7. Última resultado de coste y tiempo (mseg) para actualización global y local.

Los resultados de dichos experimentos se muestran en la Figura 5.8. En el eje horizontal se han representado únicamente las versiones del algoritmo con actualización local y global, ya

que del estudio anterior se decidió que era la mejor opción. En este caso, se representan los resultados en coste y tiempo (los valores medios) para el algoritmo con Divide y Vencerás con Subdivisión y parando cuando se llega a un nodo con olor mayor o igual que *u* (*LyGDivVen*) o sin parar (*LyGDivVenSinP*), y el algoritmo con Divide y Vencerás Simple con las mismas opciones que el anterior: parando cuando el nodo con olor encontrado tiene una cantidad del mismo mayor o igual que *u* (*LyGSimple*) y sin parar (*LyGSimpleSinP*). Además, se mostrará lo que ocurre con estas cuatro versiones del algoritmo cuando se realiza actualización global de la feromona cuando el camino global encontrado utiliza el olor (*ConGUpSmell*) y cuando no (*SinGUpSmell*).

Por último, al igual que se hacía en el estudio anterior, se muestran los resultados obtenidos para el primer camino encontrado, Figura 5.8 (a) y Figura 5.8 (b), y para el mejor después de dejar ejecutar 15 segundos más al algoritmo, Figura 5.8 (c) y Figura 5.8 (d).

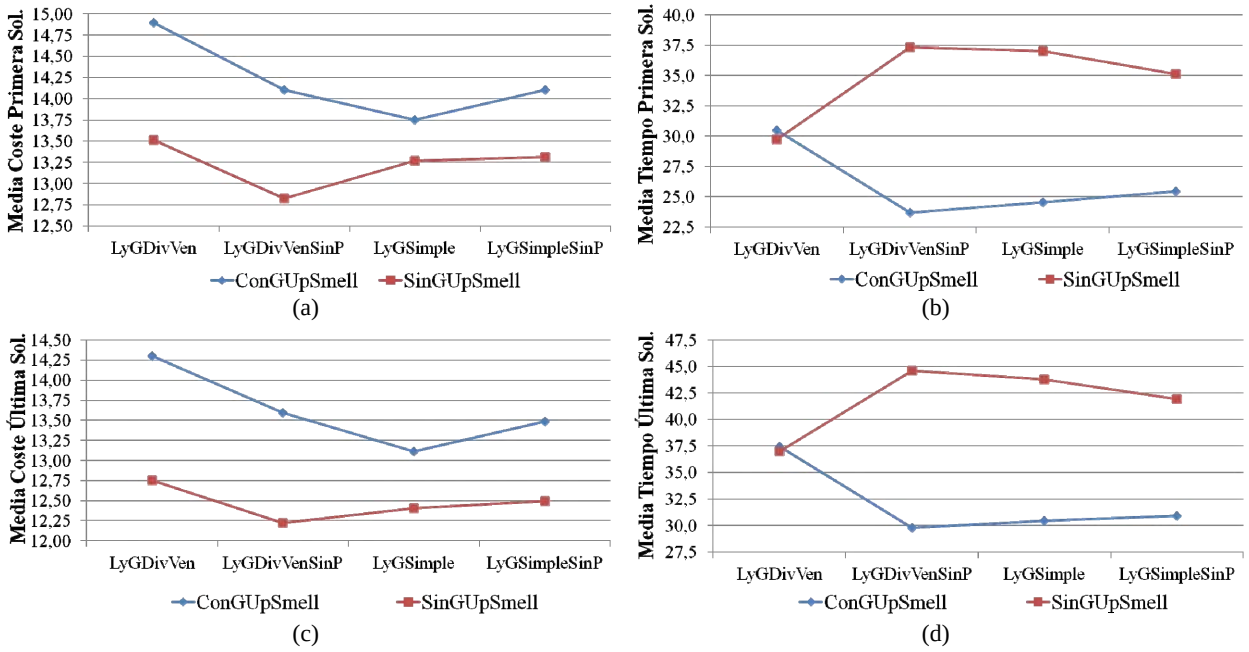


Figura 5.8. Coste y tiempo de respuesta (seg) del primer camino y del mejor camino posible dejando 15 seg de tiempo extra para las distintas versiones de división de la colonia y de actuación cuando se alcanza un olor.

Debido a que el tiempo es significativamente bueno, y a que la variación entre la peor opción y la mejor es de algo menos que 15 seg, se ha decidido atender al coste de los caminos para determinar cuál de las opciones es mejor, Figura 5.8 (a) y Figura 5.8 (c).

Como se puede apreciar en dichas figuras, el mejor de los resultados se obtiene para el caso del algoritmo con Divide y Vencerás con Subdivisión, sin actualización de feromona cuando el camino encontrado utiliza el olor como paso intermedio, y sin parar cuando encuentra un nodo con olor en el trayecto.

Este resultado parece lógico, pues el camino obtenido al pasar por el olor es una primera aproximación cuyo único fin es mostrar una solución cuanto antes y ayudar a las hormigas a

mejorarlo en posteriores búsquedas. Por este motivo, no interesa detener a una hormiga en el momento en el que encuentra s_1, sino que interesa que siga buscando para mejorar la solución que se obtiene al pasar por el área de olor. Por esta misma razón, el único efecto que se tendrá sobre la feromona de los enlaces que forman el camino obtenido, será el de la actualización local, ya que de esta manera se consigue depositar una pista de que por esos enlaces se puede alcanzar una solución, pero no se le da tanto peso a la solución como el que se le daría si la actualización fuese global, y que haría que muchas hormigas siguiesen ese trayecto reforzándolo y convirtiéndolo en el camino solución sin posibilidad de mejorarlo.

Por lo tanto, después de estudiar todas las variables que se pueden tener en cuenta cuando se dispone de un único olor a comida, el algoritmo *SoSACO* aplicará actualización local y global (esta última siempre y cuando no se utilice el nodo comida en el camino), empleará *1-tabú*, la división de la colonia denominada Divide y Vencerás con Subdivisión, una hormiga que encuentre un nodo con olor en su trayecto no reiniciará su búsqueda, y por último, el valor de *u* deberá ser elevado para que el tamaño del área con olor sea pequeño.

Dicho esto, se pasará a la fase de evaluación del siguiente ciclo del algoritmo partiendo de esta versión de *SoSACO*.

5.3.2 Grafo Estático: Varios Nodos Comida

En este apartado se partirá de la versión de *SoSACO* anterior para incluirle aquellos cambios explicados en 4.1.2 que mejores resultados aporten. Es decir, se va a proceder a realizar la experimentación pertinente para decantarse por la incorporación de unos u otros aspectos de los explicados en 4.1.2 a la versión final del algoritmo en su fase anterior. Además de esta experimentación, se añadirá una más cuyo objetivo será determinar cuál es el mejor valor para $|F|$, es decir, determinar cuántos nodos comida es interesante tener y como afecta dicho valor al comportamiento del algoritmo propuesto.

La primera comprobación que se va a llevar a cabo es la de si es interesante comprobar siempre la posibilidad de existencia de camino a través de un olor encontrado, o si es suficiente con el otro tipo de comprobación explicado en 4.1.2.1.

Para ello, los servicios ejecutados serán los mismos que los de la última tanda de experimentos del apartado anterior, es decir, tanto el nodo inicio como el destino se elegirán de manera aleatoria dentro de todos los posibles nodos del grafo.

En cuanto a los aspectos relativos a la comida y el olor, se tendrán 9 nodos comida distribuidos por el grafo y el valor de *u* será el mismo para todos, de manera que en total se tendrá un 1% de nodos con olor inicialmente en el grafo (escenario peor posible). El resto de

parámetros son los indicados en 5.2, y el algoritmo *SoSACO* se encuentra en el estado que se eligió al final del apartado anterior.

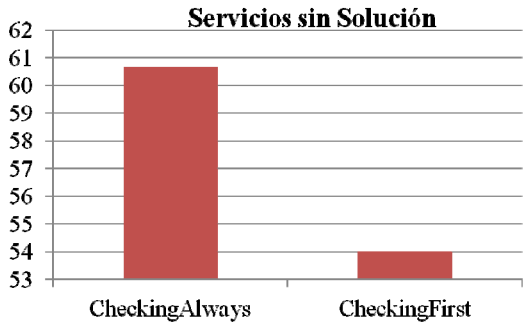


Figura 5.9. Servicios sin respuesta para varios nodos comida en función de si se comprueba siempre la existencia de camino a través del olor, o no.

Al finalizar la ejecución de los servicios, los resultados obtenidos son los que se muestran en la Figura 5.9 y en la Figura 5.10. En ellas, la opción en la que se comprueba siempre si existe o no un camino viene representada por *CheckingAlways*, y la contraria por *CheckingFirst*. Tal y como se puede apreciar en la Figura 5.10, tanto el coste como el tiempo de respuesta son mejores en el caso en el que se restringe el número de comprobaciones que se realizan. Además de la mejora en estos valores, es importante tener en cuenta que esta versión del algoritmo también mejora la tasa de éxito en la búsqueda tal y como se puede apreciar en la Figura 5.9.

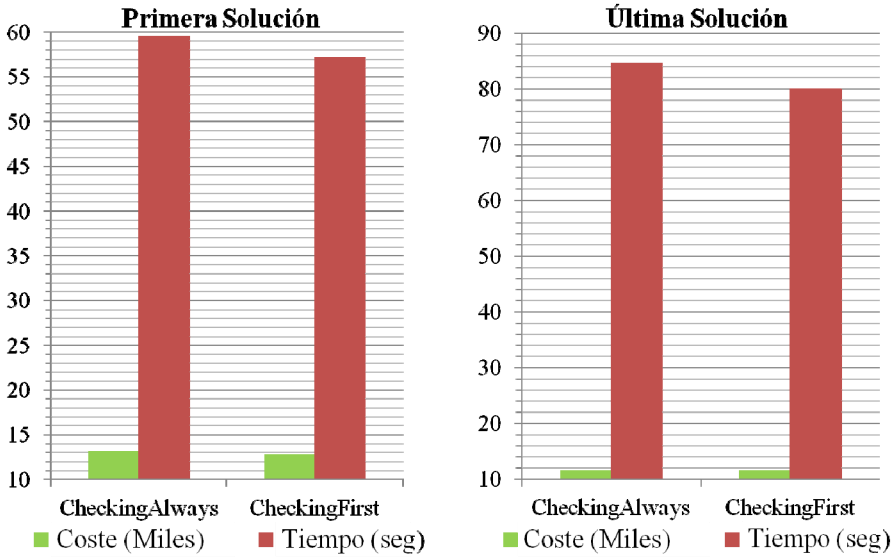


Figura 5.10. Tiempo (seg) y coste (en miles) de los servicios ejecutados cuando se tienen varios nodos comida en función de si se comprueba siempre la existencia de camino a través del olor, o no.

Este resultado era esperable, ya que esta versión del algoritmo disminuye el tiempo que es necesario emplear en comprobaciones, por lo que el tiempo de respuesta se ve reducido, pero debido a que los caminos parciales son almacenados, no se pierde la oportunidad de encontrar una solución en caso de existir. Además, está disminución del tiempo requerido para realizar comprobaciones también permitirá a las hormigas avanzar más pasos dentro del grafo, aumentando la tasa de éxito.

Por lo tanto, la mejor alternativa es emplear un número de comprobaciones restringido.

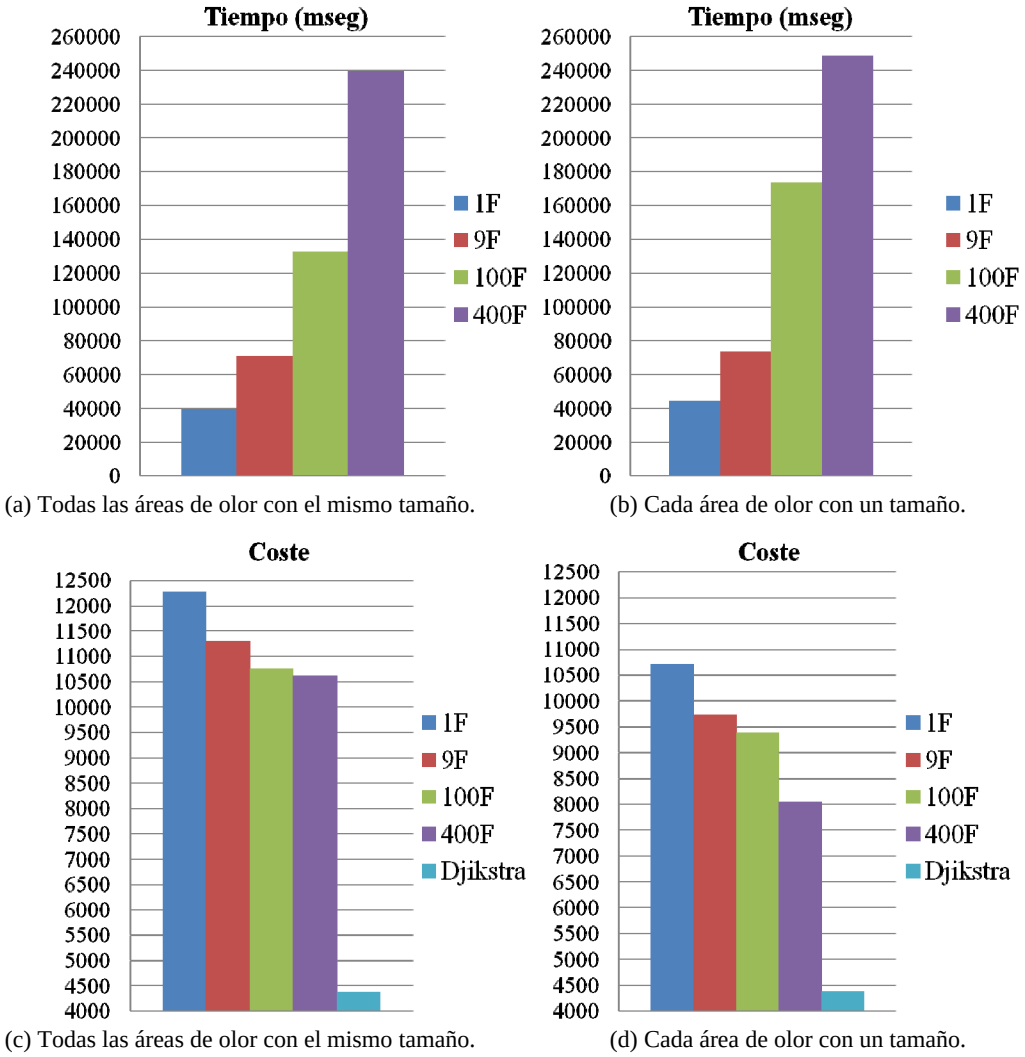
Una vez se ha decidido esto, el siguiente paso sería determinar si es mejor tener un único valor de *u*, o tener uno por cada nodo comida. Además, puesto que para ello será interesante comparar los resultados obtenidos con diversos valores de $|F|$, se podrán determinar al mismo tiempo las ventajas o desventajas de tener más o menos nodos comida distribuidos por el grafo.

Pues bien, con el fin de saber cuál era la mejor forma de usar *u*, se ejecutó *SoSACO* con todas la características decididas hasta el momento empleando un único valor de *u* que arrojaba un porcentaje del 1% de nodos con olor inicial en todo el grafo (experimentación relacionada con la primera opción planteada en 4.1.2.2), y utilizando diversos valores de umbral tal que cada uno de los nodos comida tuviese un 1% de nodos con olor (experimentación relacionada con la segunda opción planteada en 4.1.2.2). La decisión de fijar los distintos valores de umbral para tener un 1% de nodos con olor en cada f_i en la segunda opción se basaba en los resultados obtenidos en los ciclos anteriores, en los que se decidió que era mejor tener áreas de olor pequeñas. En cuanto a la primera opción, el utilizar un porcentaje de nodos con olor tan bajo en todo el grafo se debía a que, como solo se tenía un umbral, si éste se bajaba para permitir tener un mayor porcentaje, los nodos con olor se concentraban en el nodo comida con mayor grado, haciendo que el área de éste alcanzase grandes tamaños no satisfaciendo los resultados obtenidos en el primer ciclo de evolución de la propuesta.

Las difusiones de olor se realizaron sobre el grafo explicado en 5.2, y se empleó el tipo de servicios de la anterior experimentación, de hecho, la variación con respecto a ella será el número de nodos comida, de modo que habrá resultados para un único nodo comida (*1F* en la Figura 5.11), nueve nodos comida (*9F* en la Figura 5.11), cien nodos comida (*100F* en la Figura 5.11) , y cuatrocientos nodos comida (*400F* en la Figura 5.11).

En cuanto a los parámetros que se tendrán en cuenta en la experimentación serán, al igual que en los casos anteriores, el coste del camino encontrado y el tiempo requerido para obtenerlo (Figura 5.11), y el número de servicios que se han quedado sin solución al finalizar la ejecución (Figura 5.12).

Una vez dicho esto, se puede observar que existe un equilibrio en lo que respecta a coste y tiempo cuando se emplea un valor único de umbral que cuando se emplea un valor distinto para cada nodo comida. Esto es así porque el coste se ve empeorado en un 15,9% si se utiliza un valor distinto de *u* que si se emplea un único valor (Figura 5.11 (c) y (d)), pero el tiempo se ve mejorado un valor similar, en concreto un 10,28% (Figura 5.11 (a) y (b)). Sin embargo, debido a que el tiempo es el objetivo prioritario de la tesis, se utilizará un valor distinto de *u* para cada nodo comida.

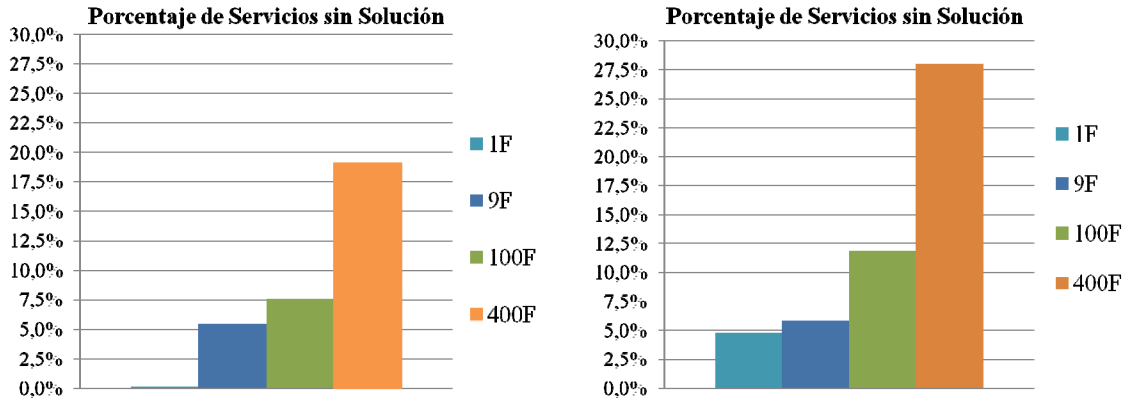


(a) Todas las áreas de olor con el mismo tamaño. (b) Cada área de olor con un tamaño.

(c) Todas las áreas de olor con el mismo tamaño. (d) Cada área de olor con un tamaño.

Figura 5.11. Tiempo y coste de los servicios ejecutados para el caso de todas las áreas de olor con el mismo número de nodos (cada área tiene su propio valor de *u*), (a) y (c), y con distintos tamaños (hay un único valor de *u*), (b) y (d).

Además, como valor añadido a la decisión tomada se debe destacar que existe una mejora del 4,56% en la tasa de éxito (Figura 5.12). Es decir, la mejor opción es la que tiene un conjunto *U*, tal que cada uno de sus elementos, u_i, indica el umbral de dispersión de olor inicial en cada nodo comida f_i.



(a) Todas las áreas de olor con el mismo tamaño. (b) Cada área de olor con un tamaño.

Figura 5.12. Porcentaje de servicios que quedan sin solución para el caso de todas las áreas de olor con el mismo número de nodos (cada área tiene su propio valor de *u*), (a), y con distintos tamaños (hay un único valor de *u*), (b).

Una vez determinado que se entiende por *u*, es necesario estudiar el comportamiento del algoritmo cuando el número de nodos comida aumenta. Para ello, se volverá a estudiar la Figura 5.11 y la Figura 5.12, pero atendiendo únicamente a aquellos gráficos correspondientes al caso elegido en la disertación anterior (Figura 5.11 (a, c) y Figura 5.12 (a)).

En lo referente al tiempo y al coste, se observa cómo mientras que el primero aumenta según lo hace el número de nodos comida (Figura 5.11 (a)) el coste hace lo contrario (Figura 5.11 (b)). La razón del incremento en tiempo se debe principalmente a que el número de comprobaciones que debe realizar cada hormiga en cada paso que da, aumenta, tardando por tanto más en encontrar un camino. Sin embargo, debido a que existen más nodos comida, la hormiga aumentará su posibilidad de encontrar uno cercano a ella, o alguno visitado por alguna hormiga del hormiguero opuesto, obteniendo de esta manera un camino inicial que ir mejorando con el paso del tiempo.

Con respecto al porcentaje de servicios sin solución (Figura 5.12 (a)), la necesidad de más tiempo por parte de las hormigas para encontrar un camino, unida a la limitación en el tiempo de búsqueda a 700 seg (Tabla 5.1), traerá consigo el que los caminos cuyos nodos extremo estén más alejados entre sí, no encuentren una solución, traduciéndose esto en un incremento de la tasa de fracaso. O lo que es lo mismo, en un descenso en la tasa de éxito.

Debido a todo lo anteriormente expuesto, se decidió que la mejor opción era elegir un valor de $|F|$ mayor que uno (para poder aprovechar la mejora en el coste), pero no elevado para tener el menor efecto posible en los otros valores. Como resultado de este análisis, los posteriores ciclos de evolución del algoritmo tomarán un valor de $\underline{|F| = 9}$.

Para terminar con este ciclo, se puede concluir que los tiempos obtenidos podrían verse mejorados con retoques en la implementación del mismo. Por ejemplo, una posible mejora vendría dada por la incorporación de comprobaciones paralelas al proceso de búsqueda, de manera que la hormiga pueda seguir buscando sin verse detenida por estos procesos y de esta manera independizar los procesos de búsqueda y comprobación.

5.3.3 Grafo Estático: Varios Nodos Comida y Caminos Almacenados

Con el fin de mejorar los resultados temporales obtenidos en el apartado anterior, y de manera secundaria tratar de mejorar el coste de las soluciones, se decidió añadir una innovación más a *SoSACO*. Esta incorporación implicaba una fase paralela a la de búsqueda del camino solicitado en la cual se buscaban los caminos entre los distintos f_i.

El hecho de tener preprocesados los caminos que unen los nodos comida va a permitir mostrar un camino rápidamente, pues solo es necesario que una hormiga de una colonia alcance un olor y una hormiga de la colonia opuesta alcance otro, o el mismo.

Este procesamiento paralelo se llevaría a cabo, idealmente, en una maquina distinta a aquella en la que se ejecutan los servicios, la búsqueda de caminos, y las alternativas planteadas son calcular caminos empleando Dijkstra o hacerlo empleando ACO con las modificaciones detalladas en el apartado 4.1.3.

Por el dinamismo propio de los grafos sobre los que quiere ejecutar el algoritmo propuesto, la mejor opción podría ser emplear ACO, no obstante, la elección realizada fue Dijkstra, ya que esta fase de la experimentación solo se desarrollará en estática, de modo que los caminos calculados entre los diversos f_i no se tendrán que volver a recalcular en ningún momento.

Una vez indicado el tipo de algoritmo empleado para realizar el preprocesado de caminos, se pasará a observar la influencia en los tres parámetros de estudio (tiempo de respuesta, tasa de éxito y calidad del camino) cuando se aplica sobre el escenario genérico indicado en 5.2 con las consideraciones especificas indicadas en el apartado anterior (Grafo Estático: Varios Nodos Comida) y con el algoritmo de búsqueda en el estado en el que se encontraba al final de dicho apartado.

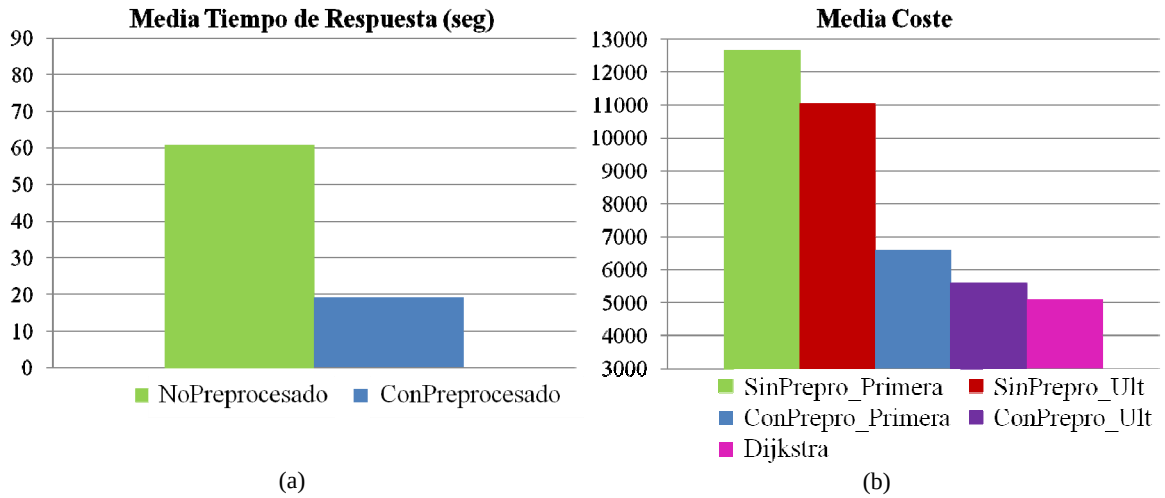
Para comenzar la experimentación en esta fase, lo primero que se hizo fue calcular los caminos existentes entre cada uno de los nueve nodos comida existentes (número de nodos comida elegido en el apartado anterior) y almacenarlos en la base de datos.

Una vez hecho esto, se procedió a ejecutar el algoritmo de búsqueda obteniendo los resultados para los parámetros de tiempo, coste y tasa de éxito que se muestran en la Figura 5.13 y la Figura 5.14.

Con respecto al tiempo de respuesta, Figura 5.13 (a), se pudo observar una mejora notable, del 68% aproximadamente, del caso en el que existe un preprocesado de caminos con respecto del caso en el que no lo hay.

A esta mejora en tiempo también hay que unir una en coste, Figura 5.13 (b), ya que el mismo se ve mejorado en un 48% en el primer camino encontrado y en un 50% en la mejor solución obtenida cuando se permite un tiempo extra de ejecución. De hecho, la diferencia con respecto a la solución óptima es del 21% en el caso del primer camino, y tan solo del 7,2% en el caso de la mejor solución encontrada. Es decir, además de mejorar el tiempo, objetivo prioritario en este trabajo de tesis doctoral, también se ha visto mejorado el coste del camino.

Del análisis anterior se puede extraer que existe una mejora en coste entre la primera solución y la mejor posible. Esto es debido a que, tal y como se fijó en la evaluación del apartado 5.3.1, cuando una hormiga encuentra un nodo con olor no reinicia su búsqueda, permitiendo de esta manera obtener una camino inicial rápidamente, pero también permitiendo mejorarlo a lo largo del tiempo.



(a) (b)

Figura 5.13. Tiempo para obtener los caminos (a) y el coste de los mismos (b) cuando se incorpora un preprocesado de caminos empleando Dijkstra.

Por último, en lo referente al porcentaje de servicios que quedan sin solución, Figura 5.14, se puede apreciar un descenso del 66%, o lo que es lo mismo, un aumento de la tasa de éxito del 34%, de cuando se tiene preprocesado de caminos a cuando no se tiene.

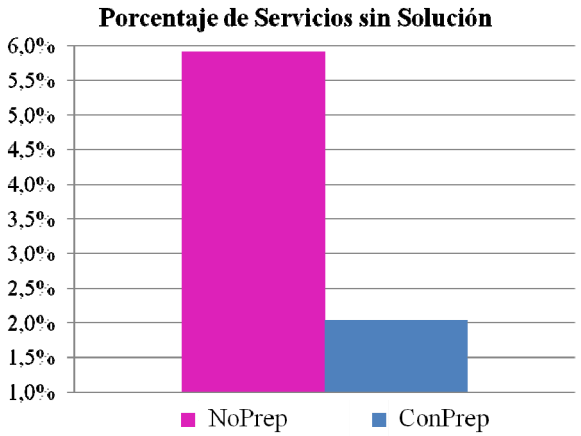


Figura 5.14. Porcentaje de servicios que quedan sin solución cuando se emplea preprocesado de caminos mediante Dijkstra.

Uniendo los resultados obtenidos en los distintos parámetros de estudio se puede concluir que el añadir este preprocesado mejora el funcionamiento del algoritmo notablemente.

5.3.4 Grafo Dinámico: Varios Nodos Comida

Tras haber ensayado en grafos vastos estáticos el algoritmo propuesto con distintas configuraciones de sus parámetros, se procede a aplicarlo sobre grafos vastos dinámicos. Con tal fin, se dispondrá de un escenario afectado por cambios con distinto periodo (T), de modo tal que el algoritmo sea evaluado con distinto grado de dinamismo. Por otro lado, en este ciclo se incorpora un nuevo parámetro a la propuesta. Se trata de la constante h, que pondera la feromona asociada a cada elemento eliminado para sopesar la necesidad de crear nuevas hormigas que desempeñen la misión de las probables hormigas *zombis* que deambulen por el grafo inútilmente a causa de esa eliminación.

En cuanto a la versión de *SoSACO* que se empleará, será el algoritmo base expuesto en el apartado 5.3.2. La mejora posterior que incluye el preprocesado de caminos entre los nodos comida en escenarios dinámicos requiere el mantenimiento de los caminos preprocesados de forma paralela a la resolución de los servicios, es decir, que debe ser realizado concurrentemente por un ACO permanente en cada nodo comida. Ese ACO buscará otros nodos comida, estabilizando las mejores soluciones y adaptándose a nuevas alternativas cuando una solución deja de ser válida. Debe observarse que este planteamiento requiere gran cantidad de cómputo. Afortunadamente, las características que proporcionan los gestores de bases de datos hacen sencilla y eficiente su distribución en un *cluster* de servidores. No obstante de la viabilidad, su coste es demasiado elevado y se dejará su evaluación y posibles mejoras para trabajos futuros.

Respecto al tipo de dinamismo que se simulará, tal y como se explicó en el apartado 4.2, consistirá en la aparición/desaparición de nodos y enlaces del grafo, a diferencia de otros trabajos estudiados en el estado del arte que exclusivamente tienen en cuenta los cambios en los costes de los enlaces.

Este dinamismo se desarrollará de forma totalmente aleatoria, lo que imposibilita incluir preprocesado orientado a los cambios que van a ocurrir (como otros trabajos en el área) ya que estos no son predecibles. Con el fin de generarlo, se escogerá un elemento del grafo cada T milisegundos (50% nodos, 50% enlaces). Todos los elementos (nodos y enlaces) tendrán un atributo (*activación*) que podrá tomar el valor 0 ó 1, de tal manera que los elementos desactivados (0) no serán tenidos en cuenta por el algoritmo (no pertenecen al grafo en el instante t). De esta manera, el cambio de estado de un enlace es una operación simple que corresponde al cambio de valor del atributo. Cuando el cambio de estado se realiza sobre un nodo deberán desactivarse todos los enlaces en los que participa, además del propio nodo.

El grafo sobre el que se ejecutará esta activación o desactivación de nodos será el explicado en el apartado 5.2, pero se le añadirán otros 200.000 nodos y 600.000 enlaces, inicialmente inactivos, con el fin de que la probabilidad de activar o desactivar un elemento sea la misma, y simular así un escenario de partida en equilibrio.

El resto de características del escenario de pruebas, así como el tipo de servicios ejecutados, serán los mismos que se explicaron en el apartado 5.3.2, con la diferencia de que en este caso se fijará $|F|$ a 9, por los resultados obtenidos en ese mismo apartado. Por otro lado, el número de hormigas que se lanzarán inicialmente será 400 para permitir que se puedan lanzar hasta 100 hormigas sustitutas, ya que el número máximo de hilos que permite la máquina sobre la que se ejecutan los experimentos es 500 (véase la Tabla 5.1).

Una vez descrito el escenario de pruebas, se pasarán a mostrar los resultados que permitirán asignar un valor para *h* así como determinar la validez de *SoSACO* en entornos dinámicos. No obstante, es importante indicar que con el fin de evitar que el proceso derivado de ejecutar el dinamismo anteriormente mencionado afectase a los tiempos de respuesta a estudiar, se han replicado los experimentos a ejecutar en dinámico en un escenario estático. Para ello, se ha ejecutado el algoritmo con dinamismo en un escenario dinámico y también sobre un grafo estático (el anteriormente descrito pero sin ejecutar los cambios), de modo que se obtiene un punto de referencia en las mismas condiciones de cómputo. Por tanto, en esta evaluación no se mostrarán valores absolutos del rendimiento del algoritmo en dinámico (por la dificultad de separar su cómputo del necesario para dotar de dinamismo al grafo), sino su comparación con respecto al rendimiento en escenarios estáticos.

En la Figura 5.15 se muestra la disminución del tiempo de respuesta de *SoSACO* en escenario dinámico con respecto al mismo algoritmo en condiciones estáticas. La evaluación se completa con el estudio del porcentaje de servicios que mejoran su tiempo de respuesta (ver Figura 5.16), cuando *T* aumenta (*T = 20, 50, 100* y *200 mseg*) para *h = 2*, *4* y *6*.

En lo referente a los tiempos de respuesta (Figura 5.15), se observa que son generalmente peores para valores elevados de *h*, siendo el mejor de los casos *h = 2*, y el peor *h = 6*. En un principio se podría pensar que los resultados iban a ser los contrarios, pues valores altos implican mayor número de hormigas sustitutas, y por tanto un mayor refuerzo en la búsqueda. Esto sería así si se pudiesen lanzar todas las hormigas sustitutas que se necesitasen, pero por estar limitado este parámetro por las especificaciones *hardware* del entorno de evaluación, las hormigas sustitutas son como máximo cien. Por este motivo, con un valor alto de *h*, se refuerzan mucho las primeras hormigas *zombi*, pero se agotan las posibilidades de refuerzo enseguida, no siendo posible aplicar refuerzo en hormigas *zombi* posteriores. Como consecuencia, las hormigas que estuvieran bien orientadas y se vean afectadas por un cambio serán completamente infructuosas, cayendo en saco roto todo su cómputo. Por el contrario, con un *h* bajo (*h = 2*) se administran mejor las posibilidades de refuerzo, manteniendo una reserva de hormigas sustitutas hasta que las *zombis* (por agotar su tiempo máximo de ejecución) empiezan a reciclarse.

Debe observarse que esta tendencia cambia según aumenta *T*, es decir, según los cambios en el grafo son menos frecuentes. En la gráfica incluso se observa que para *T = 200* se obtienen mejores resultados con *h = 4* que con *h = 2*. Esto es así porque al haber menos cambios no es necesario guardar tantas hormigas sustitutas para cambios futuros, pues no los habrá antes de que se reinicialicen las *zombis* o termine el servicio, y por tanto el algoritmo se adaptará más rápido si lanza más hormigas en cada uno de ellos.

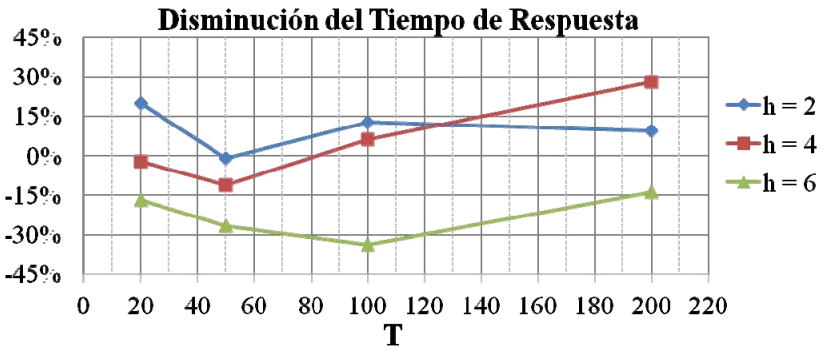


Figura 5.15. Disminución del tiempo de respuesta en escenario dinámico con respecto a escenario estático.

Por lo tanto, sin pérdida de generalidad, se puede decir que deben adaptarse los valores de *h* al grado de dinamismo, siendo proporcional al periodo de cambios en el grafo. Además, una vez fijado el valor para este parámetro, se puede observar que el tiempo de respuesta obtenido mejora hasta en un 28% con respecto a la ejecución del mismo algoritmo, con las mismas condiciones de ejecución, en estática. Esta mejoría puede entenderse mejor si se analiza el porcentaje de servicios que experimentan un descenso en su tiempo de respuesta (Figura 5.16).

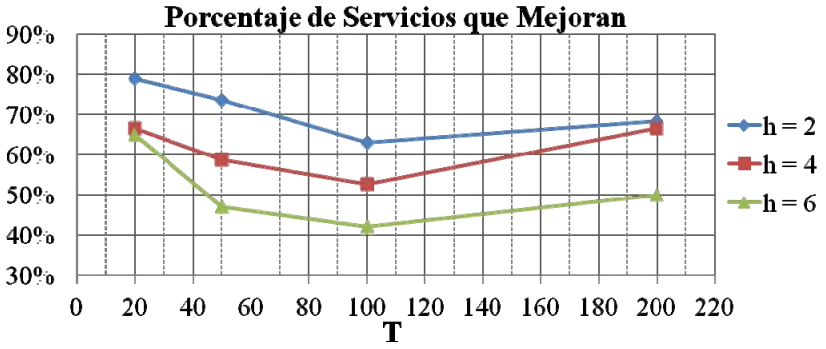


Figura 5.16. Porcentaje de servicios que mejoran cuando se aplica *SoSACO* en escenario dinámico.

En dicho porcentaje, Figura 5.16, se observa la misma tendencia en lo que respecta a *h* que en el caso anterior (el porcentaje de servicios que mejoran es menor cuanto mayor es el valor de *h*), y que el número de servicios que ven disminuido su tiempo de respuesta siempre es mayor del 60% en el caso de *h = 2*, o del 53% con *h = 4* (con *h = 6* se equilibran los casos de mejora con los que presentan peor rendimiento, que en algunos experimentos llegan a ser superiores a los de mejora en número). Esta es la causa principal de que el tiempo medio de todos los servicios ejecutados en escenario dinámico se encuentre por debajo del obtenido sobre escenario estático. No obstante, esta alta cantidad de servicios que reducen el tiempo de respuesta en escenario dinámico se ve suavizada en la Figura 5.15 debido a que el efecto de la mejora es bajo (de media un 40,6%) y constante (con una desviación típica del 8,7%), mientras que los servicios que empeoran lo hacen más drásticamente (de media 69,2%) y de una forma más variable (con una desviación típica del 28,5%).

Si además se hace un estudio de la evolución que siguen estas curvas en función de *T*, se puede observar como el porcentaje de servicios que mejoran va disminuyendo según aumenta *T*,

pareciendo apreciar cierto grado de estabilización entre *T = 40* y *T = 100*, para luego aumentar en *T = 200*. La explicación a este comportamiento se encuentra en los efectos positivos y negativos que sobre el rendimiento de *SoSACO* produce el dinamismo, y que al verse contrarrestados provocan esta tendencia, que se estabiliza posteriormente presentando una asíntota (50%) en el infinito (cuando el dinamismo es despreciable, se trata del mismo caso estático, ni mejora ni empeora).

Entre los efectos positivos del dinamismo hay que destacar que se eliminan elementos con feromona asociada no útil para la búsqueda de un servicio, disminuyendo la cantidad de posibles enlaces a utilizar (efecto de mayor peso debido a la gran cantidad de feromona espuria que se suele dar en los algoritmos ACO en sus fases iniciales, que son las que afectan a este estudio por estar dirigido a la obtención de las primeras respuestas). Por otro lado, y en menor medida, se crean nuevos atajos para la búsqueda debido a la aparición de nuevos nodos o enlaces. Con respecto a los efectos negativos de ese dinamismo, se trata de los casos opuestos: se eliminan elementos con feromona asociada útil para la búsqueda del camino; y se crean nuevos elementos superfluos que aumentan el grado de los nodos, y por tanto la probabilidad de que la hormiga se pierda. Puede completarse este análisis añadiendo que si bien estos efectos negativos son menos probables (de menor frecuencia) su influencia (cuando ocurren) es mayor.

De los resultados de la Figura 5.16 se puede concluir que tienen más peso los efectos positivos que los negativos, ya que según disminuye la frecuencia de los cambios (según aumenta *T*) disminuye el porcentaje de servicios que mejoran, tendiendo a estabilizarse a partir de *T = 100*. Es decir, el algoritmo es válido para distintos valores de *T*. De hecho, el número de servicios que se ven altamente afectados por los cambios (aquellos en los que los cambios se encuentran directamente en el camino de la hormiga sin que exista una variante directa, exigiendo un rodeo amplio) oscila solamente del 2% al 0,39% con respecto al total de servicios, y aun así, en ellos el algoritmo es capaz de encontrar una solución, aunque con un efecto negativo en el tiempo de respuesta (de media 52,3 veces mayor que el tiempo de respuesta medio de ese servicio). Es decir, la tasa de éxito se mantiene igual en el caso de dinámico que en el de estático (e incluso aumentaría si la cota $t_{threshold}$ disminuye para *h = 2* y *4*).

Debe señalarse que esto último no hubiese sido posible si se hubiese empleado Dijkstra, pues aquellos caminos que se viesen afectados por una eliminación (incluso aquellos en los que existiera una variante directa) serían invalidados teniendo que empezar la búsqueda desde el principio. De hecho, suponiendo que se parte de una situación de equilibrio en la que tanto la probabilidad de eliminar nodos/enlaces como la de crearlos es del 50% (situación de partida del escenario sobre el que se ha realizado la experimentación anterior), y teniendo en cuenta que el número medio de enlaces que tiene un nodo es 6, así como que hay la misma probabilidad de eliminar un nodo que un enlace, se tendrá que los *T* empleados (*200, 100, 50* y *20 mseg*

respectivamente) implican que el *8,3%*, *63,2%*, *87,7%* y *97,7%* de los caminos obtenidos con Dijkstra no serían válidos cuando termina el servicio..

Estos valores se pueden calcular si se tiene en cuenta la Ecuación 5.1 (a) (que se explicará a continuación) y el hecho de que el *8,3%* de los caminos está formado por *300* enlaces, el *63,2%* por *150*, el *87,7%* por *75*, y el *97,7%* por *30*.

$$T = t_{threshold} \cdot \frac{P_{EnlaceEnCamino}}{P_{EliminarEnlace}} \quad \text{(a)}$$

$$P_{EnlaceEnCamino} = \frac{EnlacesEnCamino}{|L|} \quad \text{(b)}$$

$$P_{EliminarEnlace} = P_{EliminarElemento} \cdot (P_{EliminarNodo} \cdot EnlacesNodo + P_{EliminarEnlace} \cdot 1) \quad \text{(c)}$$

Ecuación 5.1. Tiempo entre cambios para afectar a caminos.

Con respecto a la Ecuación 5.1 (a): $t_{threshold}$ es el tiempo máximo que puede ejecutar una hormiga, es decir, 700 seg según Tabla 5.1; $P_{EnlaceEnCamino}$, Ecuación 5.1 (b), indica la probabilidad de que el enlace que se borra pertenezca a un camino, y es igual al número de enlaces que forma un camino entre el número de enlaces del grafo; y $P_{EliminarEnlace}$, Ecuación 5.1 (c), indica la probabilidad de borrar un enlace, de manera que se calculará como la probabilidad de que se elimine un elemento del grafo (50%) multiplicada por la cantidad de enlaces que se borran de media (3,5 = 0,5·6 + 0,5·1).

Además, la ineficacia indicada de algoritmos como el de Dijkstra se ve agravada al existir servicios resueltos cuya solución no sea la óptima, ya que a pesar de que el camino encontrado no haya sido invalidado por las eliminaciones de elementos del grafo, podrían haber aparecido nuevos nodos/enlaces que mejorasen la solución mostrada. Dicho problema no afectaría a las hormigas, pues debido a su movimiento aleatorio, serían capaces de dar con el nuevo camino, llegando incluso, en estos casos, a dar soluciones de mejor calidad que Dijkstra.

5.4 Discusión de Resultados

Después de llevar a cabo la experimentación correspondiente a cada una de las fases consideradas en la evolución del algoritmo, se puede concluir que *SoSACO* es un algoritmo válido para ser aplicado en grafos vastos dinámicos siempre y cuando se tengan en cuenta las conclusiones que se explicarán a continuación, y que se extraen de los experimentos realizados en los distintos ciclos del algoritmo.

El tipo de actualización de la feromona depositada en cada uno de los enlaces del grafo será de tipo local y global. Esto es, la feromona se verá modificada en cada paso que dé la hormiga en su búsqueda, y cada vez que una hormiga encuentre un camino directo, sin pasar por un nodo comida, para ir del nodo inicio al destino. Esto es así porque este tipo de actualización era el que

mejores tiempos (una diferencia del 36% en media) y calidades de camino (una mejora del 15% en media) proporcionaba.

Será necesario incorporar un control sobre los nodos empleados previamente en la búsqueda para evitar, en la medida de lo posible, que aparezcan bucles en el camino. Para ello, el mejor tipo de control de la *lista tabú* será el *β-tabú*, y más concretamente el *1-tabú*, pues, aunque el coste de los caminos obtenidos era el mismo para los distintos tipos de control, teniendo en cuenta la media y la desviación típica, el tiempo era un 28% menor en este tipo que en el siguiente mejor tipo.

La colonia de hormigas inicial será dividida en dos, de tal manera que una parte irá del nodo inicio del camino al nodo destino del mismo, y la otra parte seguirá el camino opuesto. Esto será así cuando ninguno de los nodos extremo sea un nodo comida, en cuyo caso todas las hormigas partirán del nodo no perteneciente a *F*. Además, durante la ejecución, si alguna hormiga encuentra el rastro de feromona de la colonia opuesta se producirá una subdivisión. Es decir, se empleará Divide y Vencerás con Subdivisión (apartado 4.1.1.4).

Cuando una hormiga alcance un nodo con olor, es decir, cuando alcance un s_i, se procederá a buscar un camino para ir del nodo inicio al destino pasando por el f_i asociado al olor encontrado siempre y cuando no se haya marcado a dicho tipo de olor como empleado anteriormente (restricción del número de comprobaciones explicado en el apartado 4.1.2.1). El elegir este tipo de comprobación restringida se basa en que, mientras que el coste de los caminos obtenidos era el mismo que en el caso de realizar continuas comprobaciones, el tiempo de respuesta se veía mejorado en un 5,4% y el porcentaje de servicios sin solución disminuía un 6,67%.

Partiendo del caso en el que se cumplen las condiciones para comprobar si es posible encontrar un camino que vaya del nodo inicio al destino pasando por el nodo comida del tipo de olor alcanzado, y que éste existe, no se realizará la actualización global en los enlaces pertenecientes a dicho camino.

Además, en el caso de que una hormiga encuentre un nodo con olor en su trayecto, la misma no verá reiniciada su búsqueda, sino que proseguirá con ella a fin de obtener soluciones de cada vez menor coste (a igualdad de tiempo de respuesta y tasa de éxito, se mejoraba el coste en un 4% de media con respecto al caso de reiniciar la búsqueda).

Los tres comportamientos anteriores se eligieron así porque eran los que combinados aportaban un menor coste sin que el tiempo se viera afectado.

Es interesante tener zonas de olor pequeñas para evitar que una gran cantidad de cambios las afecte, así como para permitir que la recuperación en caso de verse afectadas sea rápida. Esto se decidió porque se observó que los beneficios obtenidos en casos de áreas de olor grandes con respecto a pequeñas no eran importantes comparados con el tiempo que costaba crearlas (la

diferencia entre tener áreas con el 1% de nodos con olor a áreas con el 30% incrementaba el tiempo de creación en un 98% mientras que el coste solo disminuía en un 30%). Es importante indicar que esto no limita al olor radial, ya que la forma de tratar estos rastros de olor cuando les afecta un cambio es distinta a cuando el cambio afecta a los nodos iniciales con olor (apartado 4.2.2). Es decir, la dispersión radial no afecta a la adaptabilidad de la propuesta.

El valor del umbral de olor tenía que ser distinto por cada nodo comida. Es decir, se podría elegir el tamaño de cada área de olor. La base de esta elección es la mejora en un 4,56% de la tasa de éxito obtenida en el caso de tener varios valores umbral con respecto a tener solo uno. Además, aunque el coste empeoraba en un 15,9%, el tiempo, objetivo prioritario de la tesis, mejoraba un 10,28%.

En lo referente al valor de $|F|$, se determinó que la mejor opción era disponer de pocos nodos comida. Esto fue así porque, aunque el coste disminuía según crecía el número de nodos comida, no lo hacía así el tiempo ni la tasa de éxito, de modo que como el objetivo prioritario es reducir el tiempo de respuesta, se optó por un número bajo, de manera que ni el tiempo ni la tasa de éxito se vieran altamente perjudicadas, pero que el coste se viese algo mejorado.

La incorporación de un preprocesado de caminos entre nodos comida reducía los tres parámetros de estudio drásticamente. El tiempo de respuesta disminuía en un 68% con respecto a no incluir el preprocesado, el coste en un 50%, y la cantidad de servicios sin solución en un 66%.

En lo referente al dinamismo, será interesante emplear valores de h inversamente proporcionales al grado de dinamismo (proporcionales al periodo de cambio). Así, serán bajos con cambios frecuentes para permitir que se puedan lanzar hormigas en cambios posteriores, y por tanto posibilitar una rápida adaptación para mantener el tiempo de respuesta lo más bajo posible. De hecho, en los experimentos realizados, el mejor resultado se obtenía para $h = 2$ (mínimo valor utilizado en la experimentación), que conseguía que el descenso en el tiempo de respuesta fuese del 10,29% de media.

Una vez fijados los valores para los distintos parámetros, así como elegido el tipo de comportamiento más apropiado en determinados aspectos, y tras haber estudiado los tiempos de respuesta y las calidades de los caminos encontrados en los distintos experimentos, se puede concluir que la aplicación del algoritmo es válido tanto en estática como en dinámica, pues los tiempos de respuesta se mantienen por debajo del minuto (reduciéndose hasta los 20 segundos en el caso de la incorporación del preprocesado), y las calidades se encuentran en un rango cercano al óptimo, diferenciándose únicamente en un 7,2% en el caso de añadir el cálculo de los caminos entre nodos comida.

En lo relativo a las tasas de éxito, éstas se mantienen cercanas al 100% (entre el 98% - 95%) en el caso de la aplicación del algoritmo a grafos vastos estáticos, y no experimentan variaciones en el caso de su aplicación con dinamismo, demostrando así la rápida capacidad de adaptación del algoritmo a cambios en el grafo.

5.5 Ejecución y Discusión de Resultados en Grafos Reales: Redes Sociales

Una vez demostrada la bondad del algoritmo en un entorno genérico para los distintos ciclos de evolución, se decidió observar el comportamiento del mismo en un escenario real. El objetivo principal era comprobar cuanto de bueno es el algoritmo propuesto cuando se aplica a grafos cuya estructura se supone beneficiosa para la aplicación de la aproximación propuesta.

Para ello, se eligió una red social debido a que los grafos que las representan son dinámicos, vastos, y con topología *Small-World*, que por sus características (número de enlaces para ir desde un nodo a otro relativamente bajo, ato índice de agrupamiento, y distribución del grado libre de escala) presentan un escenario de aplicación idóneo para el algoritmo propuesto, sobre todo por la primera característica mencionada, que permite reducir la cantidad de hormigas perdidas.

La red social elegida fue la presentada en [Leskovec, 2010], que corresponde a la red social Slashdot en el estado en el que se encontraba en febrero de 2009. Dicho grafo está formado por 82.168 nodos y 948.464 enlaces, y sigue una estructura de *Small-World*.

Para la captura de resultados se decidió mantener el grafo estático. El hacer esto se basó en que no se disponían de patrones de comportamiento reales en la red social de pruebas, por lo que se decidió dejar este ciclo de la experimentación como línea futura para el momento en el que se dispusiese de dicho patrón de dinamismo.

En cuanto a los algoritmos que se utilizarán en la experimentación, serán los empleados en el grafo genérico (Dijkstra y *SoSACO* en sus distintos ciclos estáticos), a los que se les añadirá el algoritmo ACO básico para ver si en este escenario, en principio beneficioso para este tipo de algoritmos, se obtienen buenos resultados. Con respecto a los parámetros con efecto en la ejecución, tomarán los valores indicados en la Tabla 5.6, cuya justificación de cada uno de ellos es la misma que la que se indicaba en el apartado 5.2 pero asociado al nuevo grafo de pruebas. De ellos, merece especial atención el valor de los distintos umbrales de olor u_i, que ha sido establecido a esa cantidad para hacer las áreas de olor lo más pequeñas posibles (solo se incluye en cada área el nodo comida y sus nodos adyacentes), satisfaciendo los resultados obtenidos en el primer ciclo de la evolución (una justificación más detallada puede encontrarse en [Rivero et al., 2011b]). Con respecto a los nodos que forman parte de *F*, indicar que se han elegido

aquellos con un mayor grado, tal que f_1 es el nodo con mayor grado, f_2 es el segundo nodo con mayor grado,... f_i es el i-ésimo nodo con mayor grado.

Parámetro	Valor
$t_{threshold}$	500 seg
#ants	500
ρ	0,6
m	1.000.000
$\forall u_i$	999.999
k	100%

Tabla 5.6. Parámetros con efecto directo en la ejecución de *SoSACO* en redes sociales.

En cuanto al número de servicios ejecutados, así como a cómo se ejecutarán los mismos, el tiempo extra que se deja ejecutar al algoritmo una vez que una hormiga ha encontrado una primera solución, y los momentos en los que reiniciar olor y feromona, serán los indicados para el caso del grafo genérico.

Respecto al tipo de servicio ejecutado, será el del apartado 5.3.3: nodo inicio y destino elegidos de manera aleatoria de entre todos los nodos posibles del grafo, tal que no haya dos servicios iguales dentro de una misma consulta.

Una vez dicho esto, las comparaciones de los cinco algoritmos se muestran en la Tabla 5.7, coste del mejor camino obtenido, la Tabla 5.8, tiempo de respuesta para obtener el mejor camino, y la Tabla 5.9, tasa de éxito. En todas estas tablas, la columna con nombre *Dijkstra* hace referencia a los resultados obtenidos cuando los servicios son resueltos por dicho algoritmo, la columna con nombre *ACO* hace referencia a los resultados obtenidos cuando se trata de dar solución a los servicios empleando el algoritmo ACO básico, y las columnas cuyos nombres empiezan por *SoSACO* muestran los resultados obtenidos con el algoritmo propuesto cuando se emplea con las características finales del apartado 5.3.1 y con un único nodo comida (*SoSACO Ciclo 1º*), con las características finales del apartado 5.3.2 y nueve nodos comida (*SoSACO Ciclo 2º*), y con las características finales del apartado 5.3.3 y nueve nodos comida (*SoSACO Ciclo 3º*).

	Dijkstra	ACO	SoSACO Ciclo 1º	SoSACO Ciclo 2º	SoSACO Ciclo 3º
Mean	4,51	387,49	5,89	5,12	4,94
SD	0,77	243,26	0,28	0,84	1,14
% SD	0,17	0,63	0,05	0,16	0,29
Median	4,00	346,75	5,88	5,00	4,00

Tabla 5.7. Coste del mejor camino obtenido al aplicar *SoSACO* a una red social.

En lo relativo al coste del camino, Tabla 5.7, se puede observar como el algoritmo propuesto obtiene un valor prácticamente igual al óptimo (el obtenido por Dijkstra). Dicha diferencia es menor según evoluciona el algoritmo a lo largo de sus ciclos, de manera que en el tercero de ellos la diferencia es prácticamente nula.

Es importante hacer notar la gran mejora que aporta el algoritmo *SoSACO* respecto al clásico, ya que, como se puede observar en la Tabla 5.7, la diferencia entre el coste obtenido en el caso del clásico comparado con cualquiera de las versiones ejecutadas del propuesto es grande, a pesar de que la topología del grafo podría ser considerada adecuada para ACO clásico debido al bajo número de enlaces que es necesario recorrer para llegar de un nodo a otro (en el peor de los casos la longitud de dicho camino son 12 enlaces según se indica en [Leskovec, 2010]).

	Dijkstra	**ACO**	**SoSACO Ciclo 1º**	**SoSACO Ciclo 2º**	**SoSACO Ciclo 3º**
Mean	563,39	253,32	12,15	16,35	19,11
SD	31,2	155,94	6,08	8,74	9,99
% SD	6%	62%	50%	53%	52%
Median	555,13	238,81	12	15,41	17,82

Tabla 5.8. Tiempo de respuesta (seg) del mejor camino obtenido al aplicar *SoSACO* a una red social.

Una vez estudiado el coste del camino obtenido, se pasa a analizar el tiempo necesario para dar una respuesta en los distintos algoritmos que se están comparando.

Al observar este parámetro en la Tabla 5.8, se hace evidente que el mejor de ellos es el algoritmo propuesto, pues sus tiempos son mucho menores que los necesarios en cualquiera de los otros casos.

En lo referente a la comparativa de tiempos de respuesta entre los distintos ciclos de evolución de la propuesta, al contrario de lo que ocurría con el coste, se observa un empeoramiento según evoluciona el algoritmo. Esto es así porque con la ayuda a la búsqueda proporcionada por el primer ciclo se tiene suficiente para mostrar un primer camino rápidamente, de modo que el efecto de las comprobaciones extra que se van incorporando ciclo tras ciclo lo que traen es un incremento en tiempo más que una disminución del mismo, que ya había alcanzado el mínimo posible con la primera versión de *SoSACO*. Esto no ocurría en la ejecución sobre el grafo genérico, ya que en dichas pruebas el número de enlaces a recorrer era elevado, y las hormigas necesitaban más ayuda para encontrar el camino, de manera que la incorporación de la misma a través de los ciclos hacía disminuir el tiempo de respuesta.

Por último, tal y como se aprecia en la Tabla 5.9, la tasa de éxito es siempre del 100% para el caso de la propuesta. Esto es importante, ya que si no se incorporase ayuda la tasa de éxito se vería reducida prácticamente un 10%, pues sería la misma que la obtenida para el caso del algoritmo ACO básico.

	Dijkstra	**ACO**	**SoSACO Ciclo 1º**	**SoSACO Ciclo 2º**	**SoSACO Ciclo 3º**
Tasa de Éxito	100%	90,6%	100%	100%	100%

Tabla 5.9. Media de la tasa de éxito al aplicar *SoSACO* a una red social.

Aunando todos estos resultados, se puede concluir que el aplicar *SoSACO* en un dominio real aporta grandes ventajas frente a los otros dos algoritmos planteados. Además, se ha observado

que las diferencias en coste vistas en la evaluación sobre grafos genéricos quedan subsanadas con la aplicación de *SoSACO* a grafos como los asociados a las Redes Sociales (grafos de topología *Small-World*).

Capítulo 6 Concluding Remarks

Once all the stages of evolution of the proposal and the analysis of the evaluation results were accomplished, some conclusions were obtained, as well as lines for future improvement.

6.1 Conclusions

At present time, most scenarios formalized with graphs require structures comprising hundreds of thousands or millions of nodes. Moreover, increasingly there are more applications working on changing structures, including to their graphs the requisite of dynamism. Finally, there are a lot of scenarios working with restrictions on response time, even as the first goal in the resolution of services, because if the response is not get in time, it would not be needed and could cause a negative impact on overall application performance. Since most of the operations performed on these graphs require the execution of a search path algorithm, it is necessary to find one which was able of satisfying all the constraints explained above.

Following a study of previous works in the state of the art, it was concluded that none of the found proposals meets all the above requirements. This can be checked in Table 6.1, which shows how, although there are studies of each of the diverse aspects mentioned, there is none that cover them jointly (a detailed explanation of this table can be found in the concluding section of the state of the art numbered 2.4).

In this table it can be seen that those works are able of providing solutions in short response time, within a lapse equal to or less than that obtained with *SoSACO*, correspond to those dealing with small graphs, or if they are huge the proposals do a global pre-processing (this feature disqualifies them for working with dynamism).

Besides, those others incorporating dynamism usually restrict the number of changes (to avoid having to do all the pre-processing from scratch), or use a dynamism previously defined (which allows to store all the possible values of the weight of the links along the time). These facts make those works inapplicable to a lot of real-world scenarios.

Finally, there are algorithms working with unlimited dynamism, generic topologies, and providing solutions in short time, amongst which the metaheuristic algorithms stand out, and more specifically thoses based on ACO, but unfortunately they are only prepared to work on small graphs.

		Huge Graph	Dynamic Graph	Generic Topology	Adapted for Secondary Storage	Pre-processing	Reduced Response Time	Path Cost
Exhaustive	Dijkstra (Chan and Lim, 2007)			√				**Optimal**
	Madduri et al., 2009 Chan and Zhang, 2001 Barrett el al., 2006	√		√		**Global**	√	**Optimal**
	Chan and Lim, 2007	√		√	√	**Global**	√	**Optimal**
	Chan and Yang, 2009		**Limited**	√		**Global**		**Optimal**
	Nannicini et al., 2010	√	**Known**			**Global**	√	**Optimal**
	Ding et al., 2008 George et al., 2007		**Known**	√	√			**Optimal**
	Bramandia et al., 2009		√	√	√	**Global**		**Optimal**
Heuristic	Delling et al., 2009 Goldberg et al., 2007	√				**Global**	√	**Optimal**
	Schultes and Sanders, 2007	√	**Known**			**Global**		**Optimal**
	Kim et al., 2007		√				√	**Reduced**
Metaheuristic	ACO (Dorigo and Blum, 2005)			√			√	**Reduced**
	Di Caro et al., 2005		√	√			√	**Reduced**
	Chicano, 2007	√		√			√	**Reduced**
	SoSACO	√	√	√	√	**Local**	√	**Reduced**

Table 6.1. Proposal within the State of the Art.

To cover that need arises *SoSACO*, which, as shown in Table 6.1, is able to provide solutions with a reduced quality, but close to optimal, using a low response time when a path search is realized in dynamic huge graphs with generic topology. That is, it has achieved a relevant impact on the existing state of the art.

With respect to the reduction of response time saving (first solution), the time to obtain the first solution is reduced to the time needed to find a food node, the same or different, by two different ant colonies, representing a considerable profit. That is due to the use of food nodes as meeting points between ants from different start nodes, and to the fact that to find them it is easy and quick using their food odor.

Regarding to the adaptability to changes that may occur in the graph, this target is well covered for joining self-adaptability of the algorithms based on ant colonies with a preprocessing (food nodes and food odor) that only requires local update, which does not affect the structure of the graph and takes little time to diffuse the odor on the graph. The fact that it

only requires local update will mean that when a structural change affects a node with food odor it was not necessary to affect all nodes with that odor unless the node affected was the food node of the odor (improbable case that does not require refresh the odor, simply delete it). In all other cases, only nodes connected to the change and having less odor than it has to be updated (that usually they will be less than half of those characterized by that odor). This involves quick adaptation to changes providing always odor areas in a consistent state. In addition, the odor is used as an aid to the ants, but not modify their basic behavior, so they do not have to stop their search process when an odor trail is being updated.

It has also been met that the rate of lost ants was reduced through the use of odor trails because the size of the graph on which the search should be performed is reduced by the fact that in areas with odor ants do not move randomly, but they move in a deterministic way following the odor trail in the increasing direction.

Finally, the cost of paths, which was considered a secondary objective, has a value within an acceptable quality range. Due to characteristics of the ACO algorithms, the obtained solutions usually are close to optimal, often providing the best solution in cost. However, and in alignment with the requirements that to this technology is done in the state of the art, the main objective is to minimize the response time. It can be stated that the evaluation of this algorithm have obtained the best response times compared with other algorithms under the same conditions, although it has finally obtained a solution which cost is close to optimal in evolved versions. That is, the ones incorporating the preprocessed paths between food nodes. It was also found that running the proposed algorithm for a longer time (after providing the first solution) provides new improved solutions (get requiring higher time costs). These conclusions are supported by the experiments performed on the test graph, which results were shown in "Capítulo 5" of this thesis documentation. In addition, some experiments on a real social network have been conducted in the same chapter, because it has a *Small-World* topology. The results are response times below the twenty seconds, a success rate of 100% and a quality almost equal to the optimum. Thus, combining experiments and their results in generic graphs and in graphs with *Small-World* topology, it can be concluded that *SoSACO* is applicable to any topology graph, as it meet the requirements exposed, and that in the case of working on real social networks its application is highly recommended.

6.2 Further Research

This work has given rise to some questions, extensions and new possibilities of application that have been left for further work. The first one regards conducting experiments on real dynamic networks, as has been done in the static stages for which social networks were used.

This would be a way of determining the behaviour of the algorithm at different dynamic scenarios, exploring variations of the algorithm to be adapted to certain cases.

Also aimed to complete the study of the behaviour of *SoSACO* in dynamic scenarios, it is interesting to build a dynamic evaluation scenario for including the Stage 3 (based on the preprocessed paths between food nodes). This scenario should include the SoSACO algorithm continuously running from each food node in order to optimise the costs of the paths found, since current results are far from optimal. Notice that, since they are supposed to be running permanently both for keeping good pheromone levels that ensure good solutions (preprocessed paths) and quick adaptation to changes (finding alternative solutions when preprocessed paths are affected), it will be necessary to have many hardware resources to attain this evaluation.

As final future work with respect to the dynamism, the initialisation of new ants created to replace *zombie ants* should be refined to make their adaptation quicker. In order to achieve this, new ants could start off from the element closest to the deleted element instead of starting off from their corresponding initial node.

One of the main extensions of the proposal consists in incorporating automated learning to the algorithm, so that the choice of food nodes and smell thresholds can be made on the basis of the experience acquired throughout the resolution of services. Therefore, new food nodes may appear, old ones disappear and areas of smell may get larger/smaller to provide better solutions when the algorithm is being executed. The food node selection policies (yet not restricted to) can include: relevant nodes (frequently used as start point or target), busy nodes (frequently visited or with high traffic), stable nodes (nodes not probably affected by changes), crossroads (nodes in more than one preprocessed path), strategic nodes (nodes far from any other food node or providing good fragmentation of graph in a given topology), and/or heuristically convenient nodes.

To avoid performance loss when the number of nodes with food marks increases too much, a dissipation threshold can be set, forcing dissipation of weaker food trails. Since this improvement depends upon the hardware configuration, it should be essayed in different configurations. In this way, food nodes and their coverage areas should be continuously inspected and balanced with performance.

Finally, to reduce the response time, it could be advantageous to include buffer of paths obtained from the requests of services. Thus, in parallel with the execution of the algorithm, a search for the existence of a path already stored could be done. Furthermore, that information could be included in the graph to bring aid to ants running at that time. In fact, any cached path could be seen as a new type of odor, serving as any food node. Therefore a new approach can be

With the addition of all these improvements, the algorithm can be improved for certain uses. Anyhow, there are several sceneries of application of diverse nature, among which stands out guiding people in Natural Interaction Systems because it was the genesis of this doctoral thesis

Natural Interaction consists in imitating human behaviour during the interaction, i.e. it reproduces the behaviour of two interacting persons. For this reason, it is important to properly manage the interruptions that the system generate when it has anything urgent to communicate, because interrupting is an aggressive action and proliferating them would bother the user. Consequently, *SoSACO* would be useful for an application of pedestrians guidance through the use of the pheromone deposited by ants on the links of the graph used to model the scenario by which the pedestrians move.

Inspecting the behaviour of *SoSACO*, it is found that when ants are looking for a destination node, they deposit pheromone on the links visited during execution. Thus, as described in this thesis documentation, the pheromone is an aid that makes possible to find the path shown as solution, marked with a large amount of pheromone (as it is reinforced because more ants use it and also because of the global update). However, given that the search contains stochastic factors, there will also be pheromone on the links next to those belonging to the solution provided, so that the amount of pheromone is greater the closer the link is to that path.

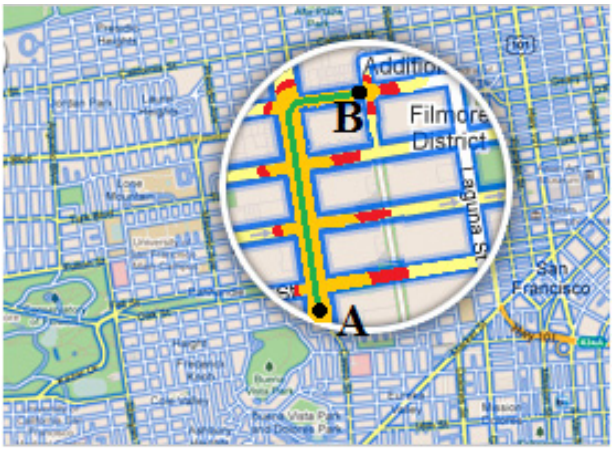


Figure 6.1. *SoSACO* in guidance of pedestrians in Natural Interaction Systems.

Consequently, when a user deviates from the marked path, the amount of pheromone associated to the last link covered by him can be used to determine the criticality of the interruption, as it will be inversely proportional to the amount of pheromone. That is, if a pedestrian is using a guidance system supported by this algorithm and wants to go from point *A* to *B* in the plane shown in Figure 6.1, the system will run *SoSACO* and it will probably obtain the path marked in green. In addition, the links near the obtained path should have pheromone, so that the links marked in amber (which are closer to the path) will have less pheromone than green links, but higher than those marked in red or yellow (the farthest, with trails close to zero). When the user moves on his way to *B*, he will not be interrupted while following the

green links, but higher than those marked in red or yellow (the farthest, with trails close to zero). When the user moves on his way to *B*, he will not be interrupted while following the green path, and he has a low probability of being interrupted if he is in the amber zone (as they are interruptions of low criticality). Conversely, he has a high probability of being interrupted when in the red zone. The measure of criticality helps to balance the benefits of interrupting (redirecting the user to increase the probability of success) with the cost of interrupting. This cost is usually variable, since it depends on the criticality of the other tasks that the user is developing at that time (either individually or jointly with the system). For example, in the red zone he can be interrupted while speaking of weather, and not interrupted in the same place if the task involves avoiding being hit by a vehicle. Through this process, a more natural interaction would be achieved because in the interaction between two persons in which one of them guides the other, when the last one is slightly wrong he is usually not corrected (he could amend his route with no extra indications), but he is redirected when deviating too much from the right path.

Also regarding guiding applications, yet focusing general interaction, the application of *SoSACO* enable working with scenarios modelled in graphs featured by a fine granularity (in which the nodes are separated from each other by a short spatial distance). That is, besides working with graphs covering large areas, the application of *SoSACO* would also enable working with huge graphs that arise from small spatial areas in which it is needed a high degree of detail. An example of this may be the application to on-line games, in which it is necessary to calculate the path to go from the node that is currently occupied by the user to the enemy (or any other element of the game), avoiding static obstacles (water, fire, traps, etc.) or mobile obstacles (other users or enemies), so it is necessary a graph that represents the state of the game in a detailed way, giving rise to a huge graph, and also allowing that the appearance of movement of the avatar was more realistic or fluid. Together with this, and because those elements endowed with movement in the game, the adaptability of the algorithm is necessary for a good performance.

Using the feature of adaptability of *SoSACO*, and related to the above explained, the moving objects networks rise as an interesting area of application. In them, the definition of the services (start and target nodes) can change, as well as the state of the network, which is dynamically altered by the effect of other moving elements in the graph. For these systems it is advisable to have an algorithm highly adaptable to changes for finding valid paths, so it is recommended the application of this proposal.

Social networks are another scenario of application of *SOSACO*, in which, as it has been shown in 5.5, the algorithm performs efficiently and effectively. In this type of applications the leaders of the graph, those most important in a particular facet [Garrido, 2010], might be

selected for placing the food nodes in them. Once this is done, *SoSACO* could be used to search a person from another using any feature of the first one, or to find the closest person that meets a particular requirement (that is, to search a not specific destination node). Besides, it also could be applied to search the person closest to the user who is an expert in a particular task (for example, in the application of ACO algorithms), or who has certain hobbies (for example, likes fantastic films). It would also support establishing relationships between affinity groups quickly (research groups in the same area).

These advantages of application would also be useful on peer-to-peer networks, in which documents with certain features are searched. In this case, food nodes could be located at nodes with a huge number of reports associated, in general, or at nodes with specific characterization in order to favour certain searches. In either of these last two scenarios there may be changes in the graph during the search of a path, so again it is advisable to apply an algorithm adaptable to changes, as is the case of *SoSACO*.

6.3 Results Dissemination

This work has given rise to some publications in national and international forums. These publications can be classified into two groups: Application of the algorithm proposed on Natural Interaction Systems, and presentation of the algorithm itself.

The first group includes the papers [Calle et al., 2008, 2010, 2011; Cuadra et al., 2008, 2009; del Valle et al., 2007, 2008, 2009, 2010; Rivero et al., 2007]. They are intended to apply the algorithm to a *Natural Interaction System*, supporting diverse tasks such as guiding in the *Situation Model*. Thus, the work presented in this thesis has an indirect influence on these items.

The second group, focused on the description of the proposed algorithm and its evaluation, includes a presentation of the proposal's general ideas [Rivero, 2009]; preliminary results [Rivero et al., 2011a]; and an assessment on social networks having a single food node [Rivero et al., 2011c]).

Apart from the publications in conferences, there are also two publications in indexed journal included in the *Journal of Citation Reports* (JCR). [Rivero et al., 2011b] explains the proposal in detail and puts forward a complete assessment of the algorithm, using both generic graphs and social networks; and [Cuadra et al., 2011] shows an study about technologies involved in the implementation of the proposal in spatio-temporal data bases. There is also now another paper under review in a journal with impact, and three papers submitted to JCR journals waiting for response.

Finally, it should be mentioned that this doctoral thesis project lies within the framework of the works carried out by the Advanced Database Group of the Universidad Carlos III of Madrid,

and in particular of the *THUBAN* (TIN2008-02711) project funded by the Ministerio de Educación y Ciencia. Nonetheless, its development has also been driven forward by other related projects, such as the *SOPAT* (CIT-410000-2007-12), *Access Channel to Digital Resources and Contents* (TSI-020501-2008-54) and *SemAnts* (TSI-020110-2009-419) projects, all of which are funded by the Ministerio de Industria, Turismo y Comercio, as well as by the *MAVIR* (S-0505/TIC-0267) and *MA2VICMR* (S2009/TIC-1542) networks subsidised by the Madrid Regional Authority. It should also be pointed out that the results of this work are being applied in the ongoing *CADOH* project, supported by the Ministerio de Industria, Turismo y Comercio.

Capítulo 7 Conclusiones

Tras llevar a cabo los distintos ciclos de evolución del algoritmo propuesto, y el análisis de los resultados obtenidos en la evaluación de cada uno de ellos para fijar los valores de los diversos parámetros con influencia en el funcionamiento del algoritmo, así como haber elegido aquellos comportamientos que mejor resultado dan en los diferentes parámetros de estudio, en este capítulo se sintetizarán los resultados más relevantes y se procederán a explicar las conclusiones extraídas de la realización de este trabajo de tesis doctoral.

7.1 Conclusiones

En la actualidad, la mayoría de los escenarios sobre los que se trabaja requieren grafos para representarlos que alcanzan tamaños de cientos de miles o millones de nodos. Además, a esto hay que añadirle el que cada vez son más las aplicaciones que trabajan sobre entornos cambiantes con el tiempo, lo que hace que los grafos que los representan adquieran cierto grado de dinamismo. Por último, se tiene que gran cantidad de estos escenarios trabajan con limitaciones en lo que a tiempo de respuesta se refiere, convirtiéndose éste en el objetivo prioritario de la resolución de servicios, pues si no se obtiene una respuesta al servicio solicitado en un tiempo fijado, la misma ya no sería necesaria y podría ocasionar consecuencias negativas en el funcionamiento global de la aplicación. Por ejemplo, en un sistema de enrutamiento de paquetes de datos, si se sobrepasa un determinado límite temporal se dará el paquete por perdido.

Puesto que la mayor parte de las operaciones que se realizan sobre estos grafos requiere de la ejecución de un algoritmo de búsqueda de caminos, es necesario encontrar uno capaz de trabajar satisfaciendo todas las limitaciones que se explicaron anteriormente.

Tras realizar un estudio de los trabajos publicados en el estado del arte (Capítulo 2), se pudo concluir que ninguna de las propuestas encontradas satisfacía la unión de todos los requisitos anteriores. Esto se puede apreciar en la Tabla 7.1, en la que se muestra como, aunque existen trabajos que cubren los distintos aspectos mencionados, no existe ninguno de ellos que los cubra

de manera conjunta (una explicación detallada de esta tabla se puede encontrar en el capítulo de Estado del Arte en el apartado de conclusiones 2.4).

		Grafo Vasto	Grafo Dinámico	Topología Genérica	Adaptado a Almacenamiento Secundario	Pre-procesado	Tiempo de Respuesta Reducido	Coste Camino
Exhaustivo	Dijkstra (Chan and Lim, 2007)			√				Óptimo
	Madduri et al., 2009 Chan and Zhang, 2001 Barrett el al., 2006	√		√		Global	√	Óptimo
	Chan and Lim, 2007	√		√	√	Global	√	Óptimo
	Chan and Yang, 2009		Restringido	√		Global		Óptimo
	Nannicini et al., 2010	√	Conocido			Global	√	Óptimo
	Ding et al., 2008 George et al., 2007		Conocido	√	√			Óptimo
	Bramandia et al., 2009		√	√	√	Global		Óptimo
Heurístico	Delling et al., 2009 Goldberg et al., 2007	√				Global	√	Óptimo
	Schultes and Sanders, 2007	√	Restringido			Global		Óptimo
	Kim et al., 2007		√				√	Reducido
Metaheurístico	ACO (Dorigo and Blum, 2005)			√			√	Reducido
	Di Caro et al., 2005		√	√			√	Reducido
	Chicano, 2007	√		√			√	Reducido
	SoSACO	√	√	√	√	Local	√	Reducido

Tabla 7.1. Propuesta dentro del Estado del Arte.

Si se hace un estudio de los trabajos que dan una solución con un tiempo de respuesta reducido, con un valor igual o menor que el obtenido con la propuesta que aquí se presenta, se puede observar que se corresponde con aquellos que tratan con grafos no vastos, o que en caso de serlo realizan un preprocesado global (hecho que los imposibilita para trabajar con dinamismo).

Además, aquellos trabajos que incorporan dinamismo lo hacen en la mayoría de los casos de manera restringida (para evitar tener que realizar todo el pre-procesado desde cero), o con cambios previamente conocidos (lo que permite tener todos los posibles valores que tomarán los costes en función del tiempo), hechos que no son aplicables a diversos escenarios del mundo real.

Respecto a aquellos algoritmos que no limitan el tipo de dinamismo, que trabajan con topologías genéricas, y que además son capaces de dar una solución en un tiempo reducido, destacan los algoritmos metaheurísticos, y más concretamente los basados en ACO. El problema es que, como se aprecia en la Tabla 7.1, trabajan sobre grafos pequeños.

Con el fin de cubrir todos estos huecos surge *SoSACO* que, como se puede apreciar en la Tabla 7.1, es capaz de dar una solución con una calidad reducida, aunque próxima a la óptima, empleando un tiempo de respuesta bajo cuando se buscan caminos dentro de grafos dinámicos vastos de topología genérica. Es decir, se ha conseguido tener un impacto de relevancia en el estado del arte existente.

En lo referente a la reducción del tiempo de respuesta del algoritmo (primera solución), debido a la utilización de los nodos comida a modo de puntos de encuentro entre hormigas procedentes desde distintos nodos inicio, y a la facilidad y rapidez para encontrarlos gracias al olor que desprenden, la búsqueda de la primera solución se reduce de media al tiempo necesario para encontrar un nodo comida, igual o distinto, por dos colonias de hormigas distintas, lo que supone un beneficio considerable.

En cuanto a la adaptación a los cambios que se puedan producir en el grafo, este objetivo queda perfectamente cubierto por haber unido la adaptabilidad propia de los algoritmos basados en colonias de hormigas con un preprocesado (nodos comida y olor a comida) que solo requiere actualización local, que no afecta a la estructura del grafo, y cuya información asociada requiere un tiempo bajo para difundirse por el grafo. El hecho de que solo requiera actualización local implicará que cuando un cambio estructural afecte a un nodo con olor a comida, no será necesario afectar a todos los nodos con ese olor, salvo que el nodo afectado sea el nodo comida del mismo (caso improbable y que no requiere refrescar el olor, sino simplemente eliminarlo). En el resto de casos, sólo se actualizarán los nodos conectados al cambio y que tengan menor olor (que habitualmente serán menos de la mitad de los caracterizados por ese olor). Esto implica una rápida adaptación a los cambios, permitiendo tener siempre a las áreas con olor en un estado consistente. Además, el olor se utiliza a modo de ayuda a las hormigas, pero no modifica su comportamiento base, de modo que éstas no tendrán que parar su proceso de búsqueda cuando se esté reiniciando un rastro de olor.

Asimismo, se ha conseguido que la tasa de pérdida de las hormigas se reduzca gracias a la utilización de los rastros de olor, pues el tamaño del grafo sobre el que se deben realizar las búsquedas se ve reducido por el hecho de que en las zonas con olor las hormigas no se mueven de manera aleatoria, sino que se mueven de manera determinista siguiendo el rastro creciente de olor a comida.

Respecto al coste de los caminos, que se había considerado un objetivo secundario, se han obtenido unos valores dentro de un rango de calidad determinado. Por las características de los algoritmos ACO, las soluciones obtenidas tendrían que ser próximas a las óptimas, proporcionando a menudo la mejor solución en coste. Sin embargo, y de forma alineada con los requerimientos que a esta tecnología se hace en el estado del arte, el objetivo principal es

minimizar el tiempo de respuesta, pudiendo afirmar que en la evaluación de este algoritmo se han obtenido los mejores tiempos de respuesta (comparando con otros algoritmos en las mismas condiciones), aunque finalmente se ha obtenido una solución cuyo coste se aproxima al óptimo en alguna de sus versiones, como en aquella en la que se incorpora un preprocesado de caminos entre los nodos comida. Además, también se ha comprobado que si se ejecuta el algoritmo de la propuesta durante más tiempo (después de aportar la primera solución), va proporcionando nuevas respuestas que sucesivamente mejoran el coste.

Todo esto ha quedado patente a lo largo de la experimentación realizada sobre el grafo de pruebas cuyos resultados se mostraban en el Capítulo 5. Así mismo, se han llevado a cabo experimentos sobre una red social real en el mismo capítulo, por poseer topología *Small-World*, que lanzan unos tiempos de respuesta inferiores a la veintena de segundos con una tasa de éxito del 100% y una calidad prácticamente igual a la óptima. De esta manera, uniendo los experimentos, y sus resultados, en grafos genéricos y grafos con topología *Small-World*, se puede concluir que *SoSACO* es aplicable a cualquier topología de grafo, y que en el caso de trabajar sobre redes sociales reales su aplicación es altamente recomendable.

7.2 Trabajos Futuros

A lo largo del desarrollo de este trabajo de tesis doctoral se han ido detectando posibles mejoras para hacer al algoritmo más eficiente y eficaz. En esta línea, a continuación se plantean diversas líneas futuras a cubrir en un periodo de tiempo de medio a corto plazo.

La primera a destacar es la de completar la fase del dinamismo llevando a cabo experimentos sobre redes reales tal y como se ha hecho en las fases estáticas, en las cuales se han empleado redes sociales. Esta sería una manera de determinar el comportamiento que tendría el algoritmo de cara a su aplicación real y comprobar si, al igual que ocurría en las fases estáticas, dicho comportamiento es óptimo bajo sus características específicas.

Además, sería interesante incluir el tercer ciclo (aquel que incorporaba un preprocesado de caminos entre los nodos comida) al dinamismo con el fin de optimizar los costes de los caminos encontrados, ya que debido a que no se puede dejar al algoritmo ejecutar todo el tiempo que necesita hasta estabilizarse, distan del valor óptimo.

Junto a esta incorporación al cuarto ciclo, también habría que añadir una experimentación que incluya un tipo de dinamismo completo: cambios estructurales del grafo y también cambios en los costes de los nodos/enlaces, ya que en la experimentación mostrada en este documento solo se tuvieron en cuenta los cambios estructurales, por ser el elemento diferenciador con respecto a otros trabajos en el área.

En lo relativo a la elección de los nodos comida, así como del umbral de olor, podría mejorar si se le incorporase un aprendizaje automático con el fin de que se realizara dicha elección en base a la experiencia que se fuese adquiriendo a lo largo de la resolución de servicios. De esta manera, durante la ejecución del algoritmo podrían aparecer nuevos nodos comida, desaparecer antiguos, o aumentar/disminuir las áreas de olor para aportar mejores soluciones gracias a una adaptación conforme a los servicios ejecutados. Es decir, por ejemplo en el caso de los nodos comida se podrían incluir nuevas políticas de selección que incluyesen a los nodos relevantes (aquellos que se utilizan frecuentemente como nodo inicio o destino), los nodos frecuentemente visitados, los nodos más estables (aquellos que tienen una baja probabilidad de verse afectados por los cambios), nodos que se encuentran en más de un camino preprocesado, nodos estratégicos (nodos alejados de otros nodos comida o que proporciona una buena fragmentación en base a la topología del grafo), y/o nodos convenientes desde un punto de vista heurístico

Otra línea futura interesante sería la de cambiar la inicialización de las nuevas hormigas creadas para sustituir a las *zombis* con el fin de que la adaptación a los cambios fuese más rápida. Para ello, las nuevas hormigas podrían partir del elemento más próximo al elemento eliminado, o creado, en lugar de partir del nodo inicio que les correspondiese.

Junto a esto, y con el objetivo de evitar una pérdida de rendimiento, ya que cuanto mayor sea el área con olor mayor es el número de cambios que le afectan perdiendo gran cantidad de tiempo evaporando olor, habría que contemplar un umbral de disipación de modo que si el número de nodos con olor crece demasiado, se disipe aquel asociado a los que tengan una menor cantidad (los menos importantes en el rastro). Es decir, se debería establecer un control continuo del número de nodos comida y del tamaño de sus áreas de olor para balancearlo con el rendimiento del algoritmo.

Ya por último, podría ser interesante almacenar en la base de datos los caminos que se vayan obteniendo durante la consecución de peticiones de servicios. De esta manera, en paralelo a la ejecución del algoritmo, se podrá realizar una búsqueda de existencia de un camino ya almacenado para disminuir el tiempo en dar una solución. Junto a esto, se podría volcar esa información sobre el grafo para aportar ayuda a las hormigas que ejecuten en ese momento. De hecho, cualquier camino almacenado podría verse como un nuevo tipo de olor, de modo que se podría dar una nueva aproximación de camino para el servicio solicitado cuando dos hormigas alcancen los nodos extremo de un mismo camino almacenado.

Con la incorporación de todas estas mejoras se podría obtener un algoritmo altamente recomendable para su aplicación en multitud de escenarios, entre los que destaca el guiado de personas dentro de Sistemas de Interacción Natural por ser la génesis del presente trabajo de tesis doctoral.

La Interacción Natural consiste en imitar el comportamiento humano durante la interacción, es decir, imitar el comportamiento que tendría una persona cuando interactúa con otra. Por este motivo, es importante gestionar de manera adecuada las interrupciones que se hacen al interlocutor por parte de la máquina cuando se le está guiando, ya que una cantidad excesiva de éstas incomodaría al usuario. Es por este motivo que *SoSACO* sería de utilidad para una aplicación de guiado a peatones gracias a la utilización de la feromona depositada por las hormigas en los enlaces del grafo que sirve para modelar el escenario por el que estos se desplazan.

En efecto, debido al funcionamiento de *SoSACO*, cuando las hormigas buscan un nodo destino van depositando feromona en los enlaces empleados durante la ejecución. De este modo, y tal como se ha descrito en esta tesis, la feromona es una ayuda que posibilita encontrar el camino mostrado como solución, marcado con gran cantidad de feromona (pues se ve reforzado por el paso de más hormigas y por la actualización global). Sin embargo, debido a que la búsqueda es aleatoria, también existirá feromona en los enlaces cercanos a los pertenecientes a la solución mostrada, siendo mayor dicha cantidad cuanto más cerca esté el enlace del camino obtenido.

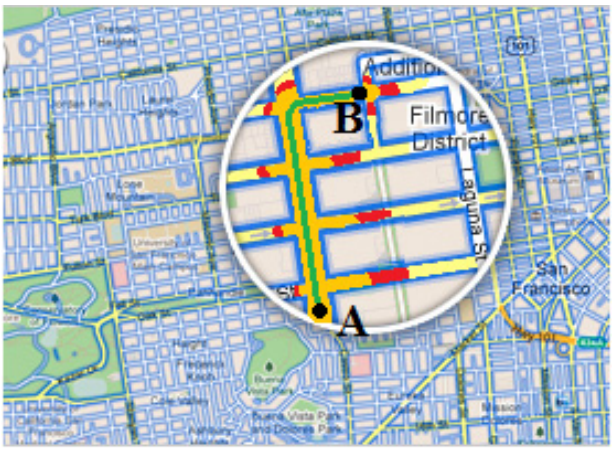


Figura 7.1. *SoSACO* en guiado de peatones en Sistemas de Interacción Natural.

En consecuencia, cuando un usuario se desvía del camino marcado, la cantidad de feromona que tenga asociado el enlace sobre el que se encuentre podrá usarse para determinar la criticidad de la interrupción, pues ésta será inversamente proporcional a la cantidad de feromona. Es decir, si un usuario que está utilizando un sistema de guiado a través de este algoritmo solicita ir del punto *A* al *B* del plano mostrado en la Figura 7.1, el sistema ejecutará *SoSACO* y obtendrá el camino marcado en verde. En los enlaces cercanos al camino obtenido también queda depositada feromona, de tal modo que los enlaces marcados en ámbar (por ser los más próximos) tendrán menor cantidad de feromona que los enlaces marcados en verde, pero mayor que los marcados en rojo o en amarillo (los más alejados, con un rastro prácticamente nulo). Cuando el usuario se mueva en su camino hacia *B*, no se verá interrumpido mientras que siga el

trayecto verde, tendrá una probabilidad muy baja de verse interrumpido si se encuentra en la zona ámbar (pues son interrupciones de baja criticidad), y una muy alta si se encuentra en la zona roja. La medida de criticidad ayudará a contrastar el beneficio de interrumpir (reorientar al usuario para aumentar la probabilidad de éxito) con el coste de interrumpir. Este coste es habitualmente variable, ya que depende de la criticidad del resto de tareas que esté desarrollando el usuario en ese momento (ya sean individualmente o conjuntamente con el sistema). Por ejemplo, en una zona roja puede ser interrumpido mientras habla del tiempo, y no interrumpido en el mismo sitio si la tarea es evitar ser atropellado por un vehículo. Gracias a todo este proceso se conseguiría una interacción más natural, ya que en la interacción entre dos personas en la que una guía a la otra, cuando ésta se equivoca levemente no se le suele corregir (puede corregirse por sí sola), pero sí se le redirige cuan se desvía demasiado del rumbo marcado.

Siguiendo con la aplicación de guiado, aunque fuera ya de los Sistemas de Interacción Natural, la aplicación de *SoSACO* permitiría trabajar con escenarios modelados en grafos que empleen una granularidad fina (en los que los nodos están separados entre ellos por una distancia espacial reducida). Es decir, además de trabajar con grafos que abarcan grandes superficies, la aplicación de *SoSACO* también permitiría trabajar con grafos vastos que surgen de áreas espacialmente pequeñas pero para las que se necesita un elevado grado de detalle. Un ejemplo de esto puede ser la aplicación a juegos on-line, en los que es necesario calcular el camino para ir desde el nodo en el que se encuentra actualmente el usuario hasta el enemigo (o cualquier elemento del juego), esquivando obstáculos estáticos (agua, fuego, trampas, etc.) o móviles (otros usuarios o enemigos), por lo que es necesario un grafo que represente el escenario de juego de una manera detallada, dando lugar a que éste sea vasto, y permitiendo además que la apariencia de movimiento del avatar sea más realista o fluido. Junto a esto, y debido a los mencionados elementos dotados de movimiento que se encuentran en la partida, la adaptabilidad del algoritmo es necesaria para obtener un buen funcionamiento.

Haciendo uso de la característica de adaptabilidad de *SoSACO*, y relacionado con lo anteriormente indicado, surgen como nuevos escenarios de aplicación las redes de objetos móviles. En ellas, los servicios pueden cambiar de definición (nodos destino e inicio), así como el estado de la red, que se ve alterado dinámicamente por el efecto de otros elementos móviles en el grafo. Para estos sistemas es recomendable contar con un algoritmo altamente adaptable a cambios para poder encontrar un camino válido, por lo que se recomienda la aplicación de esta propuesta.

Otro escenario de aplicación de *SOSACO* son las redes sociales, en las que, tal y como ha quedado patente en 5.5, se ha observado que el algoritmo se comporta de manera eficiente y eficaz. En dicho tipo de aplicaciones se podrían buscar los líderes del grafo, aquellos con mayor

importancia en una determinada característica de interés para la búsqueda [Garrido, 2010], para situar en ellos los nodos comida. Una vez hecho esto, podría emplearse *SoSACO* para buscar una persona desde otra utilizando cualquier característica de ésta, o buscar la persona más cercana que satisfaga un determinado requisito (es decir, realizar búsquedas de un nodo destino no especificado). Esto último, podría permitir buscar la persona más cercana al usuario que sea experto en una determinada tarea (por ejemplo, la aplicación de algoritmos ACO), o que tenga determinadas aficiones (por ejemplo, que le gusten las películas fantásticas). También permitiría establecer relaciones entre grupos afines (grupos de investigación en el mismo área) de una manera rápida.

Estas ventajas de aplicación también serían útiles sobre redes de tipo peer-to-peer, en las que se buscan documentos con unas determinadas características. En este caso, se podrían situar los nodos comida en nodos relacionados con mayor cantidad de informes, en general, o bien en nodos con unas características concretas para privilegiar ciertas búsquedas. En cualquiera de estos dos últimos escenarios pueden existir cambios en el estado del grafo durante la búsqueda de un camino, por lo que de nuevo es recomendable la aplicación de un algoritmo adaptable a cambios, como es el caso de *SoSACO*.

7.3 Difusión de resultados

Este trabajo ha dado lugar a publicaciones en foros nacionales e internacionales, pudiendo clasificar dichas publicaciones en dos grupos: Aplicación del algoritmo propuesto a Sistemas de Interacción Natural, y presentación del algoritmo y evaluación del mismo.

Dentro del primer grupo se encuentran los artículos [Calle et al., 2008, 2010, 2011; Cuadra et al., 2008, 2009; del Valle et al., 2007, 2008, 2009, 2010; Rivero et al., 2007]. En ellos se pretende aplicar el algoritmo a un Sistema de Interacción de Natural, permitiendo llevar a cabo diversos tipo de tareas como el guiado en el Modelo de Situación. Esto es, el trabajo presentado en esta tesis tiene una influencia indirecta en estos artículos.

En lo que al segundo grupo de trabajos se refiere, se centra más en el algoritmo propuesto, es decir, en su funcionamiento y en su evaluación. En concreto, en [Rivero, 2009] se presenta por primera vez el algoritmo reflejando sus ideas generales; en [Rivero et al., 2011a] se presentan sus primeros resultados, y en [Rivero et al., 2011c] se evalúa al algoritmo sobre redes sociales reales.

Además de las publicaciones en conferencias, también existen dos publicaciones en revistas impactadas. En [Rivero et al., 2011b] se explica en detalle la propuesta y se presenta una evaluación completa del algoritmo tanto en grafos genéricos como en redes sociales; y en [Cuadra et al., 2011] se muestra un estudio de las técnicas involucradas en la implementación de

propuesta sobre una base de datos espacio-temporal. También indicar que en la actualidad existe otro artículo en proceso de revisión en una revista también impactada (que encaja dentro del grupo de trabajos en los que se aplica la propuesta a Sistemas de Interacción Natural), y tres artículos más enviados a revistas con factor de impacto a la espera de contestación (dos del tipo anterior, y uno correspondiente a evaluaciones realizadas al aplicar *SoSACO* sobre redes sociales).

Por último, este trabajo de tesis doctoral se enmarca dentro de los trabajos del Grupo de Bases de Datos Avanzadas de la Universidad Carlos III de Madrid, y en particular del proyecto *THUBAN* (TIN2008-02711) financiado por el Ministerio de Educación y Ciencia. No obstante, su desarrollo también se ha visto impulsado por otros proyectos relacionados, como son *SOPAT* (CIT-410000-2007-12), *Canal de Acceso a Recursos y Contenidos Digitales* (TSI-020501-2008-54), *SemAnts* (TSI-020110-2009-419), y *CADOOH* (que acaba de comenzar), todos ellos financiados por el Ministerio de Industria, Turismo y Comercio. Además, se ha contado con las redes *MAVIR* (S-0505/TIC-0267) y *MA2VICMR* (S2009/TIC-1542), supeditadas al gobierno de la Comunidad de Madrid, para poder dar difusión al trabajo realizado.

Glosario

Algoritmo Paralelo
Algoritmo que puede ser dividido en partes para ejecutarse al mismo tiempo, de manera que cuando todas ellas terminen podrán unirse las soluciones de cada una y obtener un resultado final.

Analista
Rol encargado de valuar los parámetros el algoritmo.

Árbol
Grafo en el que se da el caso de que el número de caminos que conecta el nodo raíz con cualquier otro nodo es como máximo, y como mínimo, igual a uno.

del Camino Óptimo (Shortest Path Tree)
Subgrafo de un grafo dado construido a partir de los caminos óptimos entre el nodo elegido como raíz y cualquier otro nodo del grafo.

Camino
Conjunto ordenado de nodos y enlaces que es necesario recorrer para ir desde un nodo inicio a un nodo destino.

Global
Camino en el que tanto el nodo inicio como destino coinciden con el nodo inicio y destino del servicio solicitado, respectivamente.

Óptimo
Camino en el que el número de enlaces a recorrer, o la suma del coste de todos los nodos y enlaces que forman parte de él, es el mínimo posible.

Parcial
Camino que une el nodo inicio y destino de una hormiga pero en el que uno de estos dos nodos no coincide con los nodos inicio/destino del servicio solicitado.

O camino que une el nodo inicio de una hormiga (que puede coincidir o no con el nodo inicio/destino del servicio) con un nodo comida.

Consulta
Conjunto de servicios que se ejecutan de manera secuencial.

Coste

del Nodo
Valor numérico asociado al nodo que indica el peso de dicho elemento dentro del grafo al que pertenece.

del Enlace
Valor numérico asociado al enlace que indica el peso de dicho elemento dentro del grafo al que pertenece.

del Camino
Valor numérico que representa la suma de todos los costes de nodos y enlaces que forman parte de un camino.

Dinamismo
Cualquier cambio que pueda realizarse en un grafo: cambio en costes, enlaces o nodos.

Grafo
Conjunto de elementos no vacio representado por el par $\{N, L\}$ donde N es un conjunto cuyos elementos son denominados nodos y L es un subconjunto de pares no ordenados de nodos que reciben el nombre de enlaces.

Conexo
Grafo en el que entre cualquier par de nodos se puede encontrar un camino.

Dinámico
Grafo en el que se experimenta dinamismo.

Estático
Grafo cuya definición se mantiene invariable con el tiempo.

Dirigido
Grafo en el que se establece un sentido único para recorrer los enlaces, de manera que dado cualquier enlace l_{ij} se tendrá que i será su nodo inicio y j su nodo destino.

No Dirigido
Grafo en el cual $\forall l_{ij}, l_{ji} \in L$, se tiene que $l_{ij} = l_{ji}$.

No Ponderado
Grafo en el que ni los nodos ni los enlaces tienen asociado un coste.

Plano
Grafo que puede ser dibujado sobre un único plano sin que ninguno de sus enlaces se cruce salvo en sus nodos.

Ponderado
Grafo en el que los nodos y/o los enlaces tienen asociado un coste.

Vasto
Grafos cuyo tamaño hace imposible la utilización de algoritmos clásicos (exhaustivos o heurísticos) para la resolución de servicios debido a que el tiempo de respuesta necesario es inaceptable incluso en los dominios de aplicación menos estrictos en lo que a tiempo de respuesta se refiere (los que tienen como usuarios a humanos).

En este trabajo se considerará un tamaño mínimo de cientos de miles o millones de nodos.

Enlace
Relación existente entre dos nodos del grafo.

Grado de un Nodo
Parámetro que indica la cantidad de enlaces de los que forma parte un nodo.

Hormiga Zombi
Hormiga ejecutando la búsqueda de un camino que tiene como parte del camino que ha construido hasta el momento un nodo/enlace que ya no existe.

Jerarquía
Forma de estructurar un grafo de manera que éste se dividirá en niveles en los que los más altos contendrán a los inferiores.

Lista Tabú
Lista en la que se almacenan los nodos utilizados durante la búsqueda de un camino y cuyo objetivo es controlar que no se vuelvan a utilizar en un futuro.

Mínimo Local
Camino cuyo coste será el menor de aquellos próximos a él, pero que dista del óptimo.

Nodo
Unidad fundamental del grafo.

Adyacente
Nodo alcanzable de manera directa, mediante la utilización de un único enlace, desde otro nodo. O lo que es lo mismo, nodo que comparte enlace con otro nodo del grafo.

Con olor
Nodo que tiene asociado un valor de olor mayor que cero.

Comida
Nodo elegido para ser el centro de la dispersión del olor.

Destino
Nodo que forma parte de un servicio. Su característica principal dentro de éste es ser el nodo que se desea alcanzar en el camino solicitado en el servicio.

Extremo del Camino
Nodo inicio/destino del servicio solicitado.

Inicio
Nodo que forma parte de un servicio. Será desde este nodo desde el que se iniciará la búsqueda de un camino.

Intermedio
Nodo en el que coincide por primera vez la colonia de hormigas que parte del nodo inicio del servicio con la que parte del nodo destino.

Raíz
Nodo que forma parte de un árbol y que no tiene sucesores.

Visitado
Nodo que ha sido empleado con anterioridad en el proceso de resolución del servicio.

Olor
Característica numérica asociada a los nodos y que pretende indicar cuanto de lejos se está del nodo comida.

Radial
Olor cuyo valor se encuentra por debajo de un determinado umbral fijado por el analista, pero que es mayor que cero.

Inicial
Olor cuyo valor es mayor o igual a un determinado umbral fijado por el analista.

Preprocesado

Global
Tratamiento realizado sobre el grafo en el que se realizarán las búsquedas de caminos con el fin de acelerar éstas, y que implica un cambio total en la estructura del mismo debido a su división en subgrafos, a la creación de árboles, jerarquías, etc.

Esto significará que los algoritmos de búsqueda que se vayan a aplicar tendrán que estar adaptados al preprocesado realizado, y que no podrán ejecutar mientras que éste no haya concluido.

Local
Tratamiento realizado sobre el grafo en el que se realizarán las búsquedas de caminos con el fin de acelerar éstas, y que implica un cambio parcial del grafo que en ningún caso afecta a la estructura del mismo, estando más relacionada con el añadir una serie de información extra sobre él.

Puesto que no se reestructura el grafo los algoritmos de búsqueda no necesitarán que se termine el preprocesado para poder ser ejecutados y proporcionar un resultado.

Servicio
Petición de una búsqueda de camino.

Tasa

de Éxito
Porcentaje de servicios que han obtenido respuesta dentro de un margen de tiempo fijado por el analista con respecto al total de servicios solicitados.

de Pérdida de las Hormigas
Porcentaje de hormigas que una vez superado el tiempo máximo de ejecución no han alcanzado el destino.

Esto suele ocurrir cuando las hormigas tratan de solucionar un servicio dentro de grafos vastos, ya que el número de posibles nodos a los que

moverse aumenta pudiendo derivar en que la dirección elegida aleje a la hormiga cada vez más del nodo destino.

Tiempo

de Preprocesado
Tiempo empleado en realizar una restructuración del grafo, o en incorporar una información extra al grafo, que será empleada después durante la resolución de un servicio.

de Respuesta
Tiempo empleado para dar una primera solución al servicio.

Usuario
Rol encargado de solicitar los servicios al algoritmo. Puede ser un humano, otra aplicación, etc.

Bibliografía

H. Abdallah, H.M. Emara, H.T. Dorrah and A. Bahgat (2009) Using Ant Colony Optimization algorithm for solving project management problems. Expert Systems With Applications 36 (6), pp. 10004-10015.

L. Adamic and E. Adar (2005) How to search a social network. Social Networks 27 (3): 187-203.

M.A. Ahmad and J. Srivastava (2008) An Ant Colony Optimization Approach to Expert Identification in Social Networks. Social Computing, Behavioral Modeling, and Prediction, pp. 120-128.

D. Ajwani, R. Dementiev and U. Meyer (2006) A computational study of external-memory BFS algorithms. In Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, Miami, Florida, pp. 601-610.

D. Ajwani, U. Meyer and V. Osipov (2007) Improved external memory BFS implementation. In Proceedings of theWorkshop on Algorithm Engineering and Experiments, pp. 3-12.

E. Alba and F. Chicano (2007a) ACOhg: Dealing with huge graph. In: Proceedings of the Genetic and Evolutionary Computation Conference of 2007, pp. 10-17.

E. Alba and F. Chicano (2007b) Una Versión de ACO para Problemas con Grafos de muy Gran Extensión. Quinto Congreso de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2007), Puerto de la Cruz, Tenerife, España, pp. 741-748.

E. Alba, G. Leguizamón and G. Ordóñez (2007) Two models of parallel ACO algorithms for the minimum tardy task problem. International Journal of High Performance Systems Architecture 1 (1), pp. 50-59.

R. Albert and A.L. Barabási (2002) Statistical mechanics of complex networks. Reviews of Modern Physics 74, pp. 47-97.

D. Angus and T. Hendtlass (2005) Dynamic Ant Colony Optimisation. Applied Intelligence 23 (1): 33-38.

K. Appel and W. Haken (1977) Solution of the four color map problem. Scientific American 237 (4), pp. 108-121.

K. Appel, W. Haken and J. Koch (1977) Every planar map is four colourable. Illinois Journal of Mathematics 21, pp. 439-567.

D. Azar and J. Vybihal (2011) An ant colony optimization algorithm to improve software quality prediction models: Case of class stability. Information and Software Technology 53 (4), pp. 388-393.

T. Bäck, D. Fogel and Z. Michalewicz (1997) Handbook of Evolutionary Computation.

A.L. Barabási and E. Bonabeau (2003) Scale-Free Networks. Scientific American 288 (5), pp. 60–9.

C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. Marathe and D. Wagner (2006) Implementations of Routing Algorithms for Transportation Networks. DIMACS Workshop on Shortest-Path Challenge.

H. Bast, S. Funke, D. Matijevic, P. Sanders P and D. Schultes (2007) In transit to constant time shortest-path queries in road networks. In: Proceedings of Workshop on Algorithm Engineering and Experiments of 2007.

R. Bellman (1958) On a routing problem. Quarterly Applied Mathematics 16 (1), pp. 87-90.

S. Boccaletti, V. Latora, Y. Moreno, M. Chavez and D.U. Hwang (2006) Complex networks: Structure and dynamics. Physics Reports 424, pp. 175-308.

S.P. Borgatti (2005) Centrality and network flow. Social Networks 27, pp. 55-71.

R. Bramandia, J. Cheng, B. Choi and J.X. Yu (2009) Optimizing updates of recursive XML views of relations. VLDB Journal 18 (6), pp. 1313-1333.

R. Bramandia, B. Choi and W.K. Ng (2008) On incremental maintenance of 2-hop labeling of graphs. In: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), pp. 845-854.

B. Bullnheimer, G. Kotsis and C. Strauss (1999) An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 89, pp. 319-328.

F.J. Calle, E. Albacete, D. del Valle, J. Rivero and D. Cuadra (2010) Cognos: A Natural Interaction Knowledge Management Toolkit. Natural Language Processing and Information Systems, LNCS 5723/2010, pp. 303-304.

F.J. Calle, E. Albacete, G. Olaziregi, E. Sánchez, D. del Valle, J. Rivero and D. Cuadra (2011) Cognos: A Pragmatic Annotation Toolkit for the Acquisition of Natural Interaction Knowledge. To appear in 27º congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural, Huelva, Spain.

F.J. Calle, D. del Valle, J. Rivero and D. Cuadra (2008) Methodological Approach for Pragmatic Annotation. 24º congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural 41, pp. 209-216.

E. Cantú-Paz (2000) Efficient and Accurate Parallel Genetic Algorithms, chapter 7. Migration, Selection Pressure, and Superlinear Speedups, pp. 97-120.

N.K. Cauvery and K.V. Viswanatha (2008) Enhanced Ant Colony Based Algorithm for Routing in Mobile Ad Hoc Network. World Academy of Science, Engineering and Technology 46, pp. 30-35.

E.P.F. Chan and H. Lim (2007) Optimization and evaluation of shortest path queries. VLDB Journal 16 (3), pp. 343-369.

E.P.F. Chan and Y. Yang (2009) Shortest Path Tree Computation in Dynamic Graphs. IEEE Transactions on Computers 58 (4), pp. 541-557.

E.P.F. Chan and N. Zhang (2001) Finding Shortest Paths in Large Network Systems. In Proceedings of the ninth ACM international symposium on Advances in geographic information systems, Atlanta, GA, USA, pp. 160-166.

E.P.F. Chan and J. Zhang (2007) A fast unified optimal route query evaluation algorithm. In: Proceedings of the 16th ACM conference on Conference on information and knowledge management, pp. 371-380.

E.P.F. Chan and J. Zhang (2009) Efficient Evaluation of Static and Dynamic Optimal Route Queries. Advances in Spatial and Temporal Databases. Lecture Notes in Computer Science 5644, pp. 386-391.

R-S Chang, J-S Chang and P-S Lin (2009) An ant algorithm for balanced job scheduling in grids. Future Generation Computer Systems 25 (1): 20-27.

L. Chen, J. Shen, L. Qin and H. Chen (2003) An Improved Ant Colony Algorithm in Continuous Optimization. Journal of Science and Systems Engineering 12 (2), pp. 224-135.

C.H. Chen and C.J. Ting (2010) Applying Two-Stage Ant Colony Optimization to Solve the Large Scale Vehicle Routing Problem. Journal of the Eastern Asia Society for Transportation Studies 8, pp. 761-776.

J. Cheng, J.X. Yu, X. Lin, H. Wang and P.S. Yu (2008) Fast computing reachability labelings for large graphs with high compression rate. In Proceedings of the 11th International Conference on Extending Database Technology (EDBT 2008), Nantes, France, pp. 193-204.

F. Chicano (2007) Metaheurísticas e Ingeniería del Software. Doctoral Thesis. Directed by E. Alba Torres, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga.

F. Chicano and E. Alba (2008a) Ant colony optimization with partial order reduction for discovering safety violations in concurrent models. Information Processing Letters 106 (6), pp. 221-231.

F. Chicano and E. Alba (2008b) Testing Concurrent Software with Ants. ERCIM News 75, Special theme: Safety-Critical Software, pp. 30-31.

F. Chicano and E. Alba (2009) Ant Colony Optimization in Model Checking. Taller de apoyo a la decisión en Ingeniería del Software, SISTEDES 3(1), San Sebastián, Spain, pp. 25-36.

E. Cohen, E. Halperin, H. Kaplan and U. Zwick (2002) Reachability and distance queries via 2-hop labels. In Proceedings of the 13th annual ACMSIAM symposium on Discrete algorithms (SODA 2002), San Francisco, California, pp. 937-946.

T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein (2001a) Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill. ISBN 0-262-03293-7. Section 22.3: Breath-first search, pp 531 - 539.

T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein (2001b) Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill. ISBN 0-262-03293-7. Section 22.3: Depth-first search, pp.540–549.

T.G. Crainic and M. Toulouse (2003) Handbook of Metaheuristics, chapter Parallel Strategies for Metaheuristics, pp. 475-513.

D. Cuadra, J. Calle and J. Rivero (2011) How to manage spatio-temporal events in Relational Databases. JRPIT Journal of Research and Practice in Information Technology. To appear.

D. Cuadra, F.J. Calle, J. Rivero and D. del Valle (2009) Applying Spatio-Temporal Databases to Interaction Agents. International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008), Advances in Soft Computing Series 50, pp. 536-540.

D. Cuadra, J. Rivero, D. del Valle and F.J. Calle (2008) Enhancing Natural Interaction with Circumstantial Knowledge. International Transactions on Systems Science and Applications 4 (2), pp. 122-129.

S.M. De Oliveira (2009) A study of pheromone modification strategies for using ACO on the dynamic vehicle routing problem. In: Doctoral Symposium on Engineering Stochastic Local Search Algorithms of 2009, pp. 6-10.

D. del Valle, F.J. Calle, D. Cuadra and J. Rivero (2009) Breaking of the Interaction Cycle: Independent Interpretation and Generation for Advanced Dialogue Management. Human-Computer Interaction, Novel Interaction Methods and Techniques, LNCS 5611/2009, pp. 674-683.

D. del Valle, J. Rivero, F.J. Calle and D. Cuadra (2007) Conocimiento Circunstancial en Sistemas de Interacción Natural. Actas del Segundo Simposio sobre Computación Ubicua e Inteligencia Ambiental (UCAmI 2007), pp. 177-184.

D. del Valle, J. Rivero, F.J. Calle and D. Cuadra (2010) Applying a Methodological Approach to the Development of a Natural Interaction System. Organizational, Business, and Technological Aspects of the Knowledge Society, Communications in Computer and Information Science Series 112, pp. 381-386.

D. del Valle, J. Rivero, D. Conde, G. Olaziregi, J. Moreno, F.J. Calle and D. Cuadra (2008) Plataforma de Interacción Natural para el Acompañamiento Virtual. Revista Española para el procesamiento del lenguaje natural (SEPLN) 294 (41), pp. 293-294.

D. Delling, M. Holzer, K. Müller, F. Schulz and D. Wagner (2006a) High-performance multi-level graphs. 9th DIMACS Implementation Challenge, pp. 52-65.

D. Delling, M. Holzer, K. Müller, F. Schulz and D. Wagner (2009) High-performance multi-level routing. Series in Discrete Mathematics and Theoretical Computer Science 74: 73-92.

D. Delling, P. Sanders P, D. Schultes and D. Wagner (2006b) Highway hierarchies star. The Shortest Path Problem: Ninth DIMACS Implementation Challenge, DIMACS Book 74: 141-174.

G. Di Caro, F. Ducatelle and L.M. Gambardella (2005) AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. European Transactions on Telecommunications, Special Issue on Self Organization in Mobile Networking 16 (5), pp. 443-455.

J. Diesner, T.L. Frantz and K.M. Carley (2005) Communication networks from the enron email corpus "it's always about the people. enron is no different". Comput. Math. Organ. Theory 11(3), pp. 201–228.

R. Diestel (2006) Graph theory. Graduate texts in mathematics 173, ISBN 3540261834, 9783540261834.

E.W. Dijkstra (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1, pp. 269–271.

B. Ding, J.X. Yu and L. Qin (2008) Finding time-dependent shortest paths over large graphs. In proceedings of the 11th international conference on Extending database technology: Advances in database technology, Nantes, France, pp. 205-216.

M. Dorigo (1992) Optimization, learning and natural algorithms. Doctoral Thesis, Dipartamento di Elettronica, Politecnico di Milano, Italy.

M. Dorigo and C. Blum (2005) Ant colony optimization theory: A survey. TCS: Theoretical Computer Science 344, pp. 243-278.

M. Dorigo and G. Di Caro (1998) AntNet: Distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research 9, pp. 317-365.

M. Dorigo, G. Di Caro and L. Gambardella (1999) Ant Algorithms for Discrete Optimization. Artificial Life 5 (3), pp. 137-172.

M. Dorigo and L. Gambardella (1996) A study of some properties of Ant-Q. IV International C_onference on Parallel Problem from Nature, Berlin, Germany, pp. 656-665.

M. Dorigo and L. Gambardella (1997) Ant Colony System: A cooperative Learning Approach to the Travelling Salesman Problem. IEEE Transactions on Evolutionary Computation 1(1), pp. 53-66.

M. Dorigo, V. Maniezzo and A. Colorni (1991) Ant System: An Autocatalytic Optimizing Process. Technical report 91-016, Politecnico di Milano (IT), Dipartimento di Elettronica ed Informatica.

M. Dorigo, V. Maniezzo and A. Colorni (1996) The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, MAn, and Cybernetics-Part B 26(1), pp. 1-13.

M. Dorigo and T. Stützle (2003) Handbook of Metaheuristics. International Series In Operations Research and Management Science 57, chapter 9, The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, pp. 251-285.

M. Dorigo and T. Stützle (2004) Ant colony optimization. The MIT Press.

S.N. Dorogovtsev and J.F.F. Mendes (2004) The shortest path to complex networks. In: N. Johnson, J. Efstatgiou and F. Reed-Tsochas, Complex Systems and Interdisciplinary Science. World Scientific, Singapore.

N. Edmonds, A. Breuer, D. Gregor and A. Lumsdaine (2006) Single-Source Shortest Paths with the Parallel Boost Graph Library. The Ninth DIMACS Implementation Challenge: The Shortest Path Problem, Piscataway, NJ, pp. 219-248.

M. Erwig (2000) The graph Voronoi diagram with applications. Networks 36 (3), pp. 156-163.

L. Euler (1736) Solutio problematis ad geometriam situs pertinentis. Commentarii academiae scientiarum Petropolitanae 8, pp. 128-140.

C.J. Eyckelhof and M. Snoek (2002) Ant Systems for a Dynamic TSP - Ants Caught in a Traffic Jam. In: Ant Algorithms - Third International Workshop, ANTS 2002, Lecture notes in Computer Science 2462/2002, Brussels, Belgium. pp. 88-99.

H. Fan, Z. Hua, J.J. Li and D. Yuan (2004) Solving a shortest path problem by ant algorithm. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics 5, pp. 3174-3177.

S. Favuzza, G. Graditi and E. Sanseverino (2006) Adaptive and Dynamic Ant Colony Search Algorithm for Optimal Distribution Systems Reinforcement Strategy. Applied Intelligence 24 (1), pp. 31-42.

G. Feng, C. Li, Q. Gu, S. Lu and D. Chen (2006) SWS: Small World Based Search in Structured Peer-to-Peer Systems. In: Proceedings of the International Conference on Grid and Cooperative Computing Workshops of 2006, pp. 341-348.

R.W. Floyd (1962) Algorithm 97: Shortest Path. Communications of the ACM 5 (6), pp. 345.

L. R. Ford, Jr (1956) Network flow theory. Technical Report P-923, RAND, Santa Monica, CA, August, 14.

L. Forlizzi, R. H.t Güting, E. Nardelli and M. Schneider (2000) A data model and data structures for moving objects databases. Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pp.319-330.

L.C. Freeman (1979) Centrality in networks: I. Conceptual clarification. Social Networks 1, pp. 215–239.

Y. Gajpal and P.L. Abad (2009) Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. European Journal of Operational Research 196 (1), pp. 102-117.

L. Gambardella and M. Dorigo (1995) Ant-Q: A reinforcement learning approach to the traveling salesman problem. 12th International Conference on Machine Learning, San Francisco, pp. 252-260.

L.M. Gambardella and M. Dorigo (2000) An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. INFORMS Journal on Computing 12 (3), pp. 237-255.

M. Garey and D. Johson (1979) Computers and Intractibility: A Guide to the Theory of NP-Completeness. San Francisco, W.H. Freeman.

G. Garrido Yuste (2010) Community Structure in Endorsement Social Networks. M.Sc. Thesis, Departamento de Lenguajes y Sistemas Informáticos, Universidad Nacional de Educación a Distancia, Spain.

B. George, S. Kim, and S. Shekhar (2007) Spatio-temporal network databases and routing algorithms: A summary of results. In International Symposium on Spatial and Temporal Databases, pp. 460–477.

B. George and S. Shekhar (2006) Time-Aggregated Graphs for Modeling Spatio-temporal Networks. ER (Workshops)'06, pp. 85-99.

F. Glover (1977) Heuristics for integer programming using surrogate constraints. Decision Sciences 8, pp. 156-166.

F. Glover (1986) Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, pp. 533-549.

F. Glover (1998) A template for Scatter Search and Path Relinking. In J.K. Hao et al., Artificial Evolution, number 1363 in Lecture Notes in Computer Science, pp. 13-54.

F. Glover and G. Kochenberger (2002) Handbook of Metaheuristics.

F. Glover and M. Laguna (1997) Tabu search.

A.V. Goldberg, H. Kaplan and R.F. Werneck (2007) Better landmarks within reach. Ninth DIMACS Implementation Challenge: The Shortest Path Problem.

M. Gómez González (2008) Sistema de generación eléctrica con pila de combustible de óxido sólido alimentado con residuos forestales y su optimización mediante algoritmos basados en nubes de partículas, chapter 3, Técnicas Metaheurísticas. Algoritmos BAsados en Nubes de Particulas. Doctoral Thesis. Directed by F. Jurado Melguizo, Departamento de Ingeniería Eléctrica, Electrónica y de Control, Universidad Nacional de Educación a Distancia.

S. Goss, S. Aron, J.L. Deneubourg and J.M. Pasteels (1989) Self-Organized Shortcuts in the Argentine Ant. Naturwissenschft 76, pp. 579-581.

M. Guntsch and M. Middendorf (2001) Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP. Applications of Evolutionary Computing, Lecture Notes in Computer Science 2037, pp. 213-222.

M. Guntsch, M. Middendorf and H. Schmeck (2001). An Ant Colony Optimization Approach to Dynamic TSP. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO2001, pp. 860-867.

R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider and M. Vazirgiannis (2000) A foundation for representing and querying moving objects. ACM Transactions on Database Systems (TODS) 25 (1), pp.1-42.

P.E. Hart, N.J. Nilsson and B. Raphael (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2), pp. 100–107.

J.H. Holland (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.

S. Honiden, M.E. Houle, C. Sommer and M. Wolff (2010) Approximate shortest path queries using Voronoi duals. Transactions on Computational Science IX. Lecture Notes in Computer Science 6290/2010, pp. 28-53.

L.F. Ibrahim (2006) Using of Clustering and Ant-Colony Algorithms CWSP-PAM-ANT in Network Planning. International Conference on Digital Telecommunications(ICDT 2006), Cap Esterel, French Riviera, France, pp. 26-31.

L.F. Ibrahim and M.H. Al Harbi (2008) Using clustering technique M-PAM in mobile network planning. In: Proceedings of the 12th WSEAS international conference on Computers, pp. 868-873.

L.F. Ibrahim, O. Metwaly, A. Kapel and A. Ahmed (2005) Enhancing the Behavior of the Ant Algorithms to Solving Network Planning Problem. In: Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications, pp. 250-255.

J. Jaén, J.A. Mocholí, A. Catalá and E. Navarro (2011) Digital ants as the best cicerones for museum visitors. Applied Soft Computing 11 (1), pp. 111-119.

H. Jin, X. Ning amd H. Chen (2006) Efficient Search for Peer-to-Peer Information Retrieval Using Semantic Small World. In: Proceedings of the 15th international conference on World Wide Web (WWW 2006), pp. 1003-1004.

R. Jin, Y. Xiang, N. Ruan and D. Fuhry (2009) 3-HOP: A high-compression indexing scheme for reachability query. In Proceedings of the 2009 ACM SIGMOD international conference on Management of data (SIGMOD 2009), Providence, Rhode Island, USA, pp. 813-826.

R. Jin, Y. Xiang, N. Ruan and H. Wang (2008) Efficiently answering reachability queries on very large directed graphs. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD 2008), Vancouver, Canada, pp. 595-608.

D. Kadono, T. Izumi, F. Ooshita, H. Kakugawa and T. Masuzawa (2010) An ant colony optimization routing based on robustness for ad hoc networks with GPSs. Ad Hoc Networks 8 (1), pp. 63-76.

I. Katriel and U. Meyer (2003) Elementary graph algorithms in external memory. In Algorithms for Memory Hierarchies. Lecture Notes in Computer Science, 2625, New York, pp. 62–84.

H. Kautz, B. Selman and M. Shah (1997) Referral Web: combining social networks and collaborative filtering. Communications of the ACM 40 (3), pp. 63-65.

J. Kennedy (1997) The Particle Swarm: Social Adaptation of Knowledge. IEEE International Conference on Evolutionary Computation, pp. 303-308.

J. Kennedy (1999) Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1931-1938, Piscataway, NJ, USA.

J. Kennedy and R. Eberhart (1995) Particle Swarm Optimization. In: Proceedings of the IEEE International Conference on Neural Networks 4, pp. 1942-1948, Perth, Australia.

J. Kennedy and R. Eberhart (1997) A Discrete Binary Version of the Particle Swarm Algorithm. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics l5, pp. 4104-4109.

J. Kennedy, R. Eberhart and Y. Shi (2001) Swarm Intelligence. San Francisco: Morgan Kaufmann Publishers.

S. Kim, B. George and S. Shekhar (2007) Evacuation route planning: scalable heuristics. In proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, Seattle, Washington, pp. 20:1-20:8.

S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi (1983) Optimization by Simulated Annealing. Science 220 (4598), pp. 671-680.

J. Kleinberg (2000a) Navigation in a Small World. Nature 406, pp. 845.

J. Kleinberg (2000b) The small-world phenomenon: and algorithmic perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC 2000), pp. 163-170.

N. Kokash (2005) An introduction to heuristic algorithms. Research Methodology Course, Trento.

S.R. Kolavali and S. Bhatnagar (2009) Ant Colony Optimization Algorithms for Shortest Path Problems. Network Control and Optimization, Lecture Notes in Computer Science 5425, pp. 37-44.

C-Y Lee, Z-J Lee, S-W Lin and K-C Ying (2010) An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. Applied Intelligence 32 (1), pp. 88-95.

G. Leguizamón and Z. Michalewicz (1999) A new version of Ant System for subset problems. In: Proceedings of the 1999 Congress on Evolutionary Computation, Piscataway, New Jersey, USA, pp. 1459-1464.

J. Leskovec (2010) SNAP: Network datasets: Slashdot social network. Stanford University. http://snap.stanford.edu/data/soc-Slashdot0902.html. Accessed 09 November 2010.

J. Leskovec and E. Horvitz (2008) Planetary-scale views on a large instant-messaging network. In: Proceeding of the 17th international conference on World Wide Web (WWW '08), pp. 915–924, New York, NY, USA

M. Li, W.C. Lee and A. Sivasubramaniam (2004a) Semantic small world: an overlay network for peer-to-peer search. In: Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP 2004), pp. 228 - 238.

M. Li, W.C. Lee, A. Sivasubramaniam and J. Zhao (2008) SSW: A Small-World-Based Overlay for Peer-to-Peer Search. IEEE Transactions on Parallel and Distributed Systems 19 (6), pp. 735 - 749.

D. Li, X. Lu, Y. Wang and N. Xiao (2004b) Efficient Search in Gnutella-Like "Small-World" Peer-to-Peer Systems. LNCS 3032/2004, pp. 324-331.

D. Liben-Nowell (2005) An Algorithmic Approach to Social Networks. PhD thesis, MIT Computer Science and Artificial Intelligence Laboratory.

D. Liben-Nowell (2010) Wayfinding in Social Networks. Algorithms for Next Generation Networks, Computer Communications and Networks series, pp. 435-456.

H. Liu and M. Schneider (2011) Querying moving objects with uncertainty in spatio-temporal databases. Proceedings of the 16th international conference on Database systems for advanced applications.

H.R. Lourenço, O. Martin and T. Stützle (2002) Handbook of Metaheuristics, chapter Iterated local search, pp. 321-353.

G. Ma, H. Duan and S. Liu (2007) Improved Ant Colony Algorithm for Global Optimal Trajectory Planning of UAV under Complex Environment. International Journal of Computer Science & Applications 4 (3), pp. 57-68.

K. Madduri and D.A. Bader (2009) Compact graph representations and parallel connectivity algorithms for massive dynamic network analysis. In: Proceedings 23rd IEEE Int'l. Parallel and Distributed Processing Symposium (IPDPS 2009), Rome, Italy, pp. 1-11.

K. Madduri, D.A. Bader, J.W. Berry and J.R. Crobak (2009) Parallel Shortest Path Algorithms for Solving Large-Scale Instances. Ninth DIMACS Implementation Challenge: The Shortest Path Problem 74, pp. 240-290.

V. Maniezzo and A. Colorni (1999) The Ant System Applied to the Quadratic Assignment Problem. IEEE Transactions on Knowledge and Data Engineering 11 (5), pp. 769-778.

K. Mehlhorn (1988) A faster approximation algorithm for the Steiner problem in graphs. Information Processing Letters 27 (3), pp. 125-128.

K. Mehlhorn and U. Meyer (2002) External memory breadth-first search with sublinear I/O. In Proceedings of the 10th European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, 2461, New York, pp. 723–735.

B. Mendoza (2001) Uso del Sistema de la Colonia de Hormigas para Optimizar Circuitos Lógicos Combinatorios. Doctoral Thesis. Directed by C.A. Coello, Maestría en Inteligencia Artificial, Universidad Veracruzana.

U. Meyer (2001) External memory BFS on undirected graphs with bounded degree. In Proceedings of the 12th Annual Symposium on Discrete Algorithms, New York, pp. 87–88.

U. Meyer and P. Sanders (2003) Δ-stepping: a parallelizable shortest path algorithm. Journal of Algorithms 49 (1), pp. 114-152.

A. Mislove, M. Marcon, K.P. Gummadi, P. Druschel and B. Bhattachar-jee (2007) Measurement and analysis of online social networks. In: Proceedings of the 7th ACMSIGCOMM conference on Internet measurement (IMC '07), pp. 29–42, New York, NY, USA.

N. Mladenovic and P. Hansen (1997) Variable neighborhood search. Computers and Operations Research 24, pp. 1097-1100.

O. Mogren, O. Sandberg, V. Verendel and D. Dubhashi (2009) Adaptive Dynamics of Realistic Small-World Networks. In: Proceedings of the European Conference on Complex Systems 2009.

S. Molina, G. Leguizamón and E. Alba (2007) An ACO Model for a Non-stationary Formulation of the Single Elevator Problem. Journal of Computer Science and Technology, Special Issue on Selected Papers from CACiC 2006 (Argentinian Congress on Computer Science) 7 (1), pp. 45-51.

R. Montemanni, L.M. Gambardella, A.E. Rizzoli and A.V. Donati (2005) Ant Colony System for a Dynamic Vehicle Routing Problem. *Journal of Combinatorial Optimization 10* (4), pp. 327-343.

A.M. Mora, J.J. Merelo, J.L.J. Laredo, C. Millán and J. J. Torrecillas (2009) CHAC, a MOACO algorithm for computation of bi-criteria military unit path in the battlefield: presentation and first results. International Journal of Intelligent Systems 0, pp. 1-26.

H. Mühlenbein (1998) The equation for response to selection and its use for prediction. Evolutionary Computation 5, pp. 303-346.

R.J. Mullen, D. Monekosso, S. Barman and P. Remagnino (2009) A review of ant algorithms. Expert Systems with Applications 36 (6), pp. 9608-9617.

K. Munagala and A. Ranade (1999) I/O complexity of graph algorithms. In Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), New York, pp. 687–694.

A.A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea and A. Joshi (2006) On the structural properties of massive telecom call graphs: findings and implications. In: Proceedings of the 15th ACM international conference on Information and knowledge management (CIKM '06), pp. 435–444, New York, NY, USA.

G. Nannicini, P. Baptiste, G. Barbier, D. Krob and L. Liberti (2010) Fast paths in large-scale dynamic road networks. Computational Optimization and Applications 45 (1), pp. 143-158.

G. Nannicini and L. Liberti (2008) Shortest paths on dynamic graphs. International Transactions in Operational Research 15, pp. 551–563.

P. Narváez, K. Siu and H. Tzeng (2000) New Dynamic Algorithms for Shortest Path Tree Computation. ACM Transactions on Networking 8 (6), pp. 734-746.

P. Narváez, K. Siu and H. Tzeng (2001) New Dynamic SPT Algorithm Based on a Ball-and String Model. ACM Transactions on Networking 9 (6), pp. 706-718.

M.E.J. Newman (2001) Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. Physical Review E - Statistical, Nonlinear and Soft Matter Physics 64 (1), pp. 016132-1, 016132-7.

M.E.J. Newman (2003) The structure and function of complex networks. SIAM Review 45 (2), pp. 167-256.

M. Newman, A.L. Barabasi and D.J. Watts (2006) The Structure and Dynamics of networks.

I. Pohl (1971) Bi-directional Search, in B. Meltzer and D. Michie, Machine Intelligence, 6, Edinburgh University Press, pp. 127–140.

O. Quevedo (2010) Innovative Electromagnetic Designs Making Use of Periodic Structures and Advanced Optimization Tools. Doctoral Thesis. Directed by E. Rajo, Departamento de Teoría de la Señal y Comunicaciones, Universidad Carlos III de Madrid.

G. Ramalingan and T.W. Reps (1996) An Incremental Algorithm for a Generalization of the Shortest-Path Problem. Journal of Algorithms 21 (2), pp. 267-305.

G.N. Ramos, Y. Hatakeyama, F. Dong and K. Hirota (2009) Hyperbox clustering with Ant Colony Optimization (HACO) method and its application to medical risk profile recognition. Applied Soft Computing 9 (2), pp. 632-640.

G. Ren and Z. Yun (2008) Scheduling Strategy in Parallel Applications Based on Ant Colony Optimization. In: Proceedings of the 2008 International Conference on Computer Science and Software Engineering (CSSE '08) 3, pp. 82-85.

S. Ríos Insua (1996) Investigación Operativa Programación lineal y aplicaciones. Editorial Centro de Estudios Ramón Areces, S.A., pp. 225-235.

J. Rivero (2009) Fast search of paths through huge networks. In: Doctoral Symposium on Engineering Stochastic Local Search Algorithms of 2009, pp. 46-50.

J. Rivero, D. Cuadra, F.J. Calle, P. Isasi (2011a) Ant Colony to Fast Search of Paths in Huge Networks. In: Proceedings of the International Symposium on Distributed Computing and Artificial Intelligence of 2011 (DCAI), pp. 199-208.

J. Rivero, D. Cuadra, J. Calle and P. Isasi (2011b) Using the ACO algorithm for path searches in social networks. Applied Intelligence, DOI: 10.1007/s10489-011-0304-1.

J. Rivero, D. Cuadra, F.J. Calle, P. Isasi (2011c) A Bio-Inspired Algorithm for Searching Relationships in Social Networks. Proceedings of the 2011 International Conference on Computational Aspects of Social Networks (CASoN), pp. 60-65.

J. Rivero, D. Cuadra, D. del Valle and F.J. Calle (2007) Incorporating Circumstantial Knowledge Influence over Natural Interaction. IEEE International Conference on Pervasive Services, pp. 415-420.

B. Roy (1959) Transitivité et connexité. C. R. Acad. Sci. Paris 249, pp. 216–218.

S.J. Russell and P. Norvig (2003) Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2

O. Sandberg (2006) Distributed routing in small-world networks. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments, pp. 144-155.

P. Sanders and D. Schultes (2005) Highway Hierarchies Hasten Exact Shortest Path Queries. In 13th European Symposium on Algorithms (ESA), LNCS 3669, pp. 568-579.

P. Sanders and D. Schultes (2006) Robust, almost constant time shortest-path queries in road networks. In 9th DIMACS Implementation Challenge 1.

P. Sanders and D. Schultes (2007) Engineering Fast Route Planning Algorithms. In 6th Workshop on Experimental Algorithms (WEA), LNCS 4525, pp. 23-36.

J. Sankaranarayanan, H. Samet and H. Alborzi (2009) Path oracles for spatial networks. In: Proceedings of the 35th International Conference on Very Large Data Bases, pp. 1210-1221.

J. Sankaranarayanan and H. Samet (2009) Distance Oracles for Spatial Networks. IEEE 25th International Conference on Data Engineering, Shanghai, China, pp. 652-663.

R. Schenkel, A. Theobald and G. Weikum (2004) Hopi: An efficient connection index for complex XML document collections. In Proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004), LNCS 2992/2004, pp. 665-666.

R. Schenkel, A. Theobald and G. Weikum (2005) Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. In Proceedings of the 21th International Conference on Data Engineering (ICDE 2005), pp. 360-371.

R. Schoonderwoerd, J.L. Bruten, O.E. Holland and L.J.M. Rothkrantz (1996) Ant-based load balancing in telecommunications networks. Journal Adaptive Behavior 5 (2), pp. 169-207.

D. Schultes and P. Sanders (2007) Dynamic Highway-Node Routing. In 6th Workshop on Experimental Algorithms (WEA), LNCS 4525, pp. 66-79.

J. Sousa and C. Bento (2009) People as Ants: A Multi Pheromone Ubiquitous Approach for Intelligent Transport Systems. 3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008, Advances in Soft Computing series 51, pp. 321-325.

T. Stützle (1999) Local search algorithms for combinatorial problems analysis, algorithms and new applications. Technical report, DISKI Dissertationen zur Künstliken Intelligenz.

T. Stützle and H. Hoos (1998) Improvements on the Ant System: MAX-MIN Ant System. In: Proceedings of Artificial Neural Networks and Genetic Algorithms 1997, pp. 245-249.

T. Stützle and H. Hoos (2000) Max-Min Ant System. Future Generation Computer Systems 16 (9), pp. 889-914.

L. Tang and H. Liu (2010) Graph Mining Applications to Social Network Análisis. In: C. Aggarwal and H. Wang. Managing and Mining Graph Data. Advances in DataBase Systems 40, pp. 487-514.

L. Tang, H. Liu, J. Zhang and Z. Nazeri (2008) Community evolution in dynamic multi-mode networks. In: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD ’08), pp. 677–685, New York, NY, USA.

J. Travers and S. Milgram (1969) An experimental study of the small world problem. Sociometry 32 (4), pp. 425-443.

T. Walsh (1999) Search in a Small World. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), pp. 1172-1177.

C.R. Wang and Y. Li (2009) The Application of the Ant Colony Pheromones in Intelligent Learning. International Forum on Information Technology and Applications, pp. 488-491.

J. Wang, E. Osagie, P. Thulasiraman and R.K. Thulasiram (2009) HOPNET: A hybrid ant colony optimization routing algorithm for mobile ad hoc network. Ad Hoc Networks 7 (4), pp. 690-705.

S. Warshall (1962) A theorem on Boolean matrices. Journal of the ACM 9 (1), pp. 11–12.

D.J. Watts and S.H. Strogatz (1998) Collective dynamics of “small-world” networks. Nature 393(6684), pp. 440-442.

D.J. Watts (2004) The “New” Science of Networks. Annual Review of Sociology 30, pp. 243-270.

D.J. Watts, P.S. Dodds and M.E.J. Newman (2002) Identity and Search in Social Networks. Science 296, pp. 1302-1305.

Z. Xiao-hua, M. Hong-yun and J. Li-cheng (2005) Intelligent particle swarm optimization in multiobjective optimization. The 2005 IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, pp. 714 -719 (Vol.1).

J.X. Yu and J. Cheng (2010) Graph Reachability Queries: A Survey. In: C. Aggarwal and H. Wang. Managing and Mining Graph Data. Advances in DataBase Systems 40, pp. 181-215.

B. Yu, Z.-Z. Yang and B. Yao (2009) An improved ant colony optimization for vehicle routing problem. European Journal Of Operational Research 196 (1), pp. 171-176.

W. Yuan, D. Guan, Y-K Lee and S. Lee (2010) The small-world trust network. Applied Intelligence. ISSN 0924-669X, pp. 1-12.

J. Zhang, M.S. Ackerman and L. Adamic (2007) Expertise Networks in Online Communities: Structure and Algorithms. In: Proceedings of the Sixteenth International World Wide Web Conference (WWW'07), pp. 221-230.

P. Zhao and J. Han (2010) On graph query optimization in large networks. In Proceedings of the VLDB Endowment 3 (1-2), pp. 340-351.