6. ANEXOS

6.1 CONTENIDO DE LOS ANEXOS

6.2. Algoritmos de optimización basados en colonias de hormigas (ACO)

En este apartado se trata de dar una breve introducción a los conceptos introducidos por Dorigo y Stützle en el libro Ant Colony Optimization (Dorigo & Stützle., 2004), con los cuales se va a guiar en todo momento el proceso de desarrollo del programa. De esta forma todas las decisiones tomadas en el proceso de construcción del mismo podrán ser justificadas haciendo las referencias oportunas.

6.3. Desarrollo del algoritmo

Este es el apartado fundamental de la memoria. En él, tras una introducción explicativa de cómo se va a implementar el algoritmo y de cuales van a ser sus bases de desarrollo, respectivamente en los sub-apartados '6.3.1. Base de trabajo del algoritmo' y '6.3.2. Bases de desarrollo del algoritmo', se procede a explicar cada una de las modificaciones que se han ido desarrollando del programa dentro del apartado '6.3.3. Construcción del algoritmo'. Además de la ya citada explicación justificativa de cada uno de ellos, se introducen los resultados obtenidos al aplicar dicho programa a alguno de los grafos usados a tal efecto y, finalmente, se realiza un análisis de dichos resultados a fin de justificar o no la utilidad de dicho programa, marcando tanto sus puntos fuertes como los posibles inconvenientes que estos pudieran tener.

6.4. Programa definitivo

Como último apartado de los anexos, se ha colocado el programa desarrollado a día 4_4_2011, el cual, desarrollado con el lenguaje de programación Python, ha sido considerado como el programa definitivo y con el cual se obtuvieron mejores resultados, tal y como se puede observar en las conclusiones dadas en el sub-apartado *A día 4_4_2011* dentro del apartado '6.3. Desarrollo del algoritmo'.

6.2. ALGORITMOS DE OPTIMIZACIÓN BASADOS EN COLONIAS DE HORMIGAS (ACO)

Los algoritmos de optimización basados en colonias de hormigas (Ant Colony Optimization algorithms) (ACO), son utilizados tanto para reducir como para solucionar diversos problemas computacionales a través de una búsqueda aleatoria de caminos o trayectos en grafos de forma que se vayan mejorando las soluciones obtenidas previamente. Ésta familia de algoritmos se considera inmersa en el grupo de optimizaciones meta heurísticas, ya que se tratan de algoritmos aproximados de optimización y búsqueda de propósito general y, además de eso, son procedimientos iterativos que guían una heurística subordinada combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda. Para aclarar el concepto de algoritmo heurístico, éste puede ser definido como aquel que abandona uno de los dos objetivos fundamentales en computación, encontrar buenas soluciones y ejecutarse razonablemente rápido. Sin embargo, es necesario remarcar que, en caso de abandonar el segundo objetivo, no hay pruebas de que todas las soluciones vayan a ser correctas y adecuadas o de que, en caso de abandonar el primer objetivo, todas las soluciones vayan a ser ejecutadas rápidamente.

Las ventajas de usar algoritmos meta heurísticos, entre otras, es que se tratan de algoritmos de propósito general, poseen un gran éxito en la práctica y además es sencillo implementarlos y paralelizarlos. Sin embargo, esta serie de algoritmos lleva consigo ciertos inconvenientes como el hecho de que son altamente no determinísticos (probabilísticos), son aproximados, es decir, no exactos, y además presentan poca base teórica. Generalmente, las técnicas meta heurísticas se suelen aplicar a aquellos problemas que carecen de un algoritmo o heurística específica que proporcione soluciones satisfactorias o bien cuando es imposible implementar ese método óptimo. En términos generales se podrían definir las técnicas meta heurísticas como aquellas que encuentran soluciones aproximadas (no óptimas) a problemas basándose en un conocimiento anterior (a veces llamado experiencia) de los mismos.

La base teórica de ésta serie de algoritmos de optimización (ACO) se basa en un mecanismo natural observado en distintas especies de hormigas cuando éstas buscan fuentes de comida alrededor de sus colonias, tal y como se puede observar en la Figura 1. Generalmente, dichas hormigas vagan aleatoriamente en su búsqueda de comida y, tan pronto como éstas la encuentran, vuelven a la colonia dejando tras de sí un rastro de feromonas en su camino. Una vez esto ocurre, el resto de hormigas de la colonia dejan de buscar comida de manera aleatoria y comienzan a seguir el rastro dejado por las anteriores hormigas fortaleciendo a su vez dicho rastro de feromonas en el camino en el caso de que encontraran dicha fuente de comida. Las característica principal de éste rastro de feromonas es la evaporación de las mismas en función del tiempo. Debido a ello, contra más tiempo necesite una hormiga para alcanzar la comida y volver a la colonia, más tiempo tendrán las feromonas para evaporarse y, por tanto, dicho recorrido será menos atractivo para las siguientes hormigas. Por ejemplo, en el caso concreto de que una hormiga

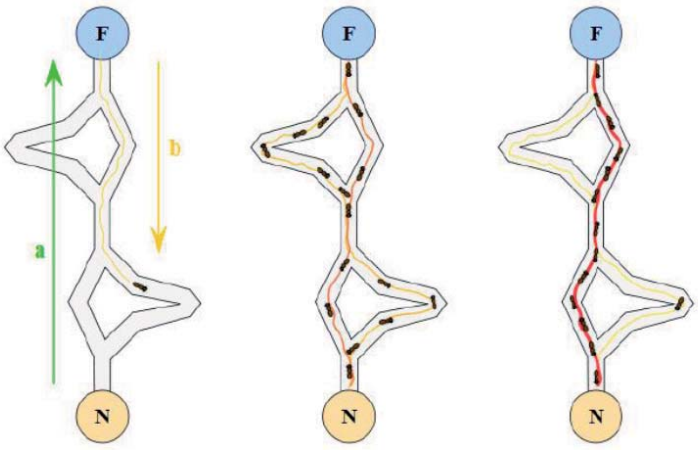

Figura 1: Explotación de los recursos de comida

encuentre un recorrido suficientemente corto entre la comida y la colonia, la densidad de feromonas en dicho camino permanecerá elevada ya que no habrá discurrido el tiempo suficiente como para que éstas se evaporen y, por tanto, dicho camino resultará mucho más atractivo para las siguientes hormigas que otros recorridos más largos y, a su vez, puesto que éste será tomado por mas hormigas, las feromonas dejadas en dicho recorrido irán en aumento, mostrando así la conveniencia del mismo. El fenómeno de evaporación de las feromonas con el tiempo también tiene la ventaja de que evita la convergencia en una solución óptima local ya que, si no existiera dicho fenómeno de evaporación, los recorridos escogidos por las primeras hormigas que encontraran una fuente de comida tenderían a ser excesivamente atractivos para las siguientes, en cuyo caso, la exploración del espacio de soluciones sería restringida.

Las dos características principales que hacen que éste sistema usado por las hormigas para intercambiar información sea distinto a otras formas de comunicación son, en primer lugar, que se trata de un sistema indirecto, no simbólico, llevado a cabo a través del medio natural y, en segundo lugar, que la información que se trasmite tiene un carácter local, es decir, tan solo pueden acceder a la información de un determinado punto del medio aquellas hormigas que discurren por él.

Los algoritmos de optimización basado en colonias de hormigas (algoritmos ACO) se tratan, como ya se comentó anteriormente, de técnicas probabilísticas para resolver aquellos tipos de problemas computacionales que pueden ser reducidos a encontrar buenos recorridos a través de grafos. El primer algoritmo de ésta familia fue inicialmente propuesto en 1992 por Marco Dorigo en su Phd tesis, (Dorigo, 1992) con el objetivo de encontrar un recorrido óptimo en un grafo mediante la imitación a través de unas "hormigas simuladas" del comportamiento real de las hormigas en su búsqueda de fuentes de comida. A partir de éste primer algoritmo, esta idea original de estudiar y aplicar el comportamiento natural de las hormigas diversificó a fin de resolver una mayor cantidad de problemas numéricos. La base en la cual Dorigo se apoyó para presentar su PhD tesis fue el ''experimento de los dos puentes'', realizado en 1990 por Deneubourg y otros (Deneubourg, Aron, Goss, & Pasteels., 1990) con el propósito de investigar el proceso de depósito de feromonas por una colonia de hormigas en su búsqueda de comida. En dicho experimento, la colonia se encontraba conectada con la fuente de comida a través de dos puentes de igual longitud. En un primer momento, se observó que cada hormiga seleccionaba uno de los dos puentes de forma aleatoria debido a una serie de fluctuaciones estocásticas pero al cabo de un tiempo uno de los dos puentes presentaba una mayor concentración de feromonas. Como consecuencia de dicha mayor concentración en uno de los dos puentes, éste atraía más hormigas que el otro y, por tanto, una mayor cantidad de feromonas eran depositadas en éste por las siguientes hormigas llevando al resultado de que, al cabo de un tiempo, toda la colonia de hormigas tomaba dicho puente en lugar que el otro. Dicho experimento fue repetido un elevado número de veces llegando a observarse que cada puente era escogido por las hormigas alrededor de un 50% de las ocasiones. La principal conclusión de éste experimento fue que las hormigas podían llegar a encontrar el recorrido más corto entre la fuente de comida y su colonia mediante el uso de la llamada ''retroalimentación positiva'', basada en que el rastro de feromonas atraía otras hormigas que, a su vez, fortalecían dicho rastro con el depósito de más feromonas.

Una variante de dicho experimento, en el cual uno de los dos puentes era significativamente más largo que el otro fue llevada a cabo por Goss y otros (Deneubourg, Aron, Goss, & Pasteels., 1989). A consecuencia de dicha diferencia de longitud entre ambos puentes, las fluctuaciones estocásticas en la selección inicial de un puente fueron reducidas y, puesto que las hormigas que casualmente escogían el puente más corto regresaban con comida a la colonia antes que aquellas que habían escogido el camino largo, el puente con menor recorrido recibía las feromonas más rápido que el largo y, como consecuencia, la probabilidad de que futuras hormigas seleccionaran éste camino más corto se incrementaban. La probabilidad p_1 de que una hormiga escogiera el

primer puente en un momento dado fue modelada de la siguiente manera:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h}$$

Ecuación 2

Donde m_1 es el número de hormigas que han usado el primer puente, m_2 el segundo y los parámetros k y h fueron determinados en función de datos experimentales. Obviamente $p_2 = 1 - p_1$.

Generalmente, el comportamiento de las hormigas en su búsqueda de comida, el cual pone las bases para los algoritmos de optimización basados en colonias de hormigas (ACO), está basado en dos sistemas opuestos de retroalimentación y puede ser modelado de la siguiente manera:

1. Una hormiga realiza recorridos más o menos de forma aleatoria alrededor de la colonia.
2. En caso de que encuentre comida, ésta retorna a la colonia más o menos directamente dejando tras de sí en el camino un rastro de feromonas.
3. Dado que dichas feromonas son atractivas para las otras hormigas, éstas seguirán más o menos el camino marcado.
4. Una vez de vuelta a la colonia, éstas nuevas hormigas fortalecerán la ruta al dejar tras de sí más feromonas.
5. Si hubiera más de un recorrido para alcanzar la misma fuente de comida, en un periodo concreto de tiempo, el camino más corto será recorrido por más hormigas que el largo.
6. El recorrido más corto irá incrementando su cantidad de feromonas al ser recorrido por más hormigas, llegando a ser cada vez más atractivo (lo cual se le denomina retroalimentación positiva).
7. El recorrido más largo, eventualmente irá desapareciendo debido al fenómeno de evaporación de las hormonas (lo cual se le denomina retroalimentación negativa).
8. Finalmente, todas las hormigas estarán determinadas a recorrer el camino más corto.

Es necesario notar que, a pesar del hecho de que ningún camino sería elegido como mejor si la cantidad de feromonas en los ejes del grafo representativo fuera idéntica a lo largo del tiempo en todos los ejes a consecuencia de las opuestas retroalimentaciones, una ligera variación en la cantidad de feromonas en uno de ellos provocará que ésta sea amplificada y por tanto, el algoritmo saldría de un estado inestable en el cual ningún camino es mejor que el otro, para pasar a un estado estable en el cual el mejor camino sería el compuesto por los mejores ejes. Como ya se ha comentado anteriormente, los algoritmos ACO se consideran mecanismos meta heurísticos para problemas de optimización combinatorios. A fin de aplicar los mismos, es básica la existencia de un modelo, el cual es descrito a continuación.

> *Un modelo $P = (S, \Omega, f)$ de un problema de optimización combinatorio consta de:*
>
> - *Un espacio de búsqueda S definido sobre un conjunto finito de variables de decisión concretas $X_i,\ i = 1, \dots, n$.*
> - *Un conjunto Ω de restricciones entre las variables.*
> - *Una función objetivo $f: S \rightarrow \mathbb{R}_o^+$ a ser minimizada.*
>
> *La variable genérica X_i toma valores en $D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$. Una posible solución $s \in S$ se trata de la completa asignación de valores a variables que satisfagan todo el conjunto de restricciones Ω. Una solución $s^* \in S$ es llamada óptimo global si y solo si $f(s^*) \leq f(s) \quad \forall s \in S$.*

Este modelo fue utilizado para definir el comportamiento de las feromonas en el algoritmo desarrollado. Un

valor de feromona es asociado con cada posible componente de solución, es decir, con cada posible asignación de un valor a una variable. Formalmente el valor de feromona τ_{ij} es asociado con el componente de solución c_{ij} , que consiste en asignar $X_i = v_i^j$. En general para este tipo de algoritmos, una ''hormiga artificial'' construye una solución atravesando el completo grafo conexo de construcción $G_c(V, E)$, donde V es el conjunto de vértices y E es el conjunto de ejes. Dicho grafo puede ser obtenido a partir del conjunto de todos los posibles componentes, llamado C, de dos maneras diferentes, ya sea mediante la representación de éstos componentes por vértices o por ejes. Al mismo tiempo, es importante denotar que las soluciones parciales son construidas de forma incremental cuando las hormigas se mueven de un vértice a otro a través de los ejes del grafo depositando en ellos una cierta cantidad de feromona. La cantidad de feromona que éstas depositan en los componentes del grafo atravesados,$\Delta\tau$, es dependiente de la calidad de la solución encontrada y posteriormente, dicha cantidad de feromona es usada por otras hormigas como una guía en el espacio de búsqueda.

Básicamente, el procedimiento de trabajo de los algoritmos meta heurísticos ACO es iterar sobre tres fases bastante bien definidas, tal y como se explica a continuación. Resumiendo, en cada iteración, un determinado número de soluciones son construidas, entonces, dichas soluciones son opcionalmente mejoradas mediante una búsqueda local y, como último paso, el valor de feromonas es actualizado.

Algoritmo meta heurístico basado en colonias de hormigas

Preparación de parámetros, Inicialización de los rastros de feromonas

mientras condición de finalización no satisfecha **hacer**:

Construcción de soluciones

Aplicar búsqueda local (opcional)

Actualizar feromonas

Fin mientras

Construcción de soluciones: Un conjunto de m 'hormigas artificiales' construyen soluciones a partir de los elementos de un conjunto finito de componentes de solución disponibles $C = \{c_{ij}\},\ i = 1, \dots, n\ \ j = 1, \dots, |D_i|$. La construcción de una solución comienza a partir de una solución parcial vacía $s^p = \emptyset$. En cada pasa del proceso de construcción, la solución parcial s^p es ampliada mediante la adición de posibles componentes de solución desde el conjunto $N(s^p) \subseteq C$, el cual es definido como el conjunto de componentes que pueden ser añadidos a la actual solución parcial s^p sin violar ninguna de las restricciones en Ω. El proceso de construcción de soluciones puede ser considerado como un recorrido en el grafo de construcción $G_c = (V, E)$. La selección de un componente de solución desde $N(s^p)$ es guiada por un mecanismo estocástico, el cual está basado, como ya se comentó anteriormente, por la cantidad de feromonas asociadas con cada elemento de $N(s^p)$. La regla para la selección estocástica de componentes para la solución varía a través de los diferentes tipos de algoritmos de optimización basados en colonias de hormigas (algoritmos ACO).

Aplicar búsqueda local: Una vez que las soluciones ya han sido construidas, y justo anteriormente a que las feromonas sean actualizadas, es común mejorar las soluciones obtenidas por las hormigas a través de una búsqueda local. Ésta fase, que varía enormemente según el tipo de problema que se trate, aun siendo opcional es altamente recomendable y generalmente es incluida en los algoritmos de optimización basados en colonias de hormigas más eficientes.

Actualizar feromonas: El propósito de la actualización de feromonas es incrementar los valores de feromona asociados con buenas o prometedoras soluciones a la vez que reduciendo aquellas que se asocian a malas soluciones. Normalmente, esto es conseguido, como ya se ha comentado anteriormente, reduciendo todos los valores de feromonas a través de una evaporación de las mismas e incrementando los niveles de las mismas en aquellas asociadas al conjunto de buenas soluciones.

Cuando el primer algoritmo ACO fue presentado por Dorigo a principios de los 90 usando ''el Problema del viajante'' como ejemplo de aplicación, dicho algoritmo, llamado 'Ant System'(sistema de hormiga) proporcionó peores resultados que los algoritmos más conocidos en aquel momento para solucionar dicho problema. A pesar de ello, los resultados proporcionados por dicho algoritmo fueron alentadores y, hoy en día, los algoritmos relacionados con la optimización basada en colonias de hormigas más exitosos son extensiones de este primer algoritmo (AS). Estas extensiones utilizan el mismo mecanismo de actualización de los valores de feromonas que realizaba el primer algoritmo pero añadiendo tan solo algunas simples modificaciones.En un principio, todo el trabajo desarrollado alrededor de los diferentes algoritmos de optimización basados en colonias de hormigas eran desarrollados experimentalmente con el propósito de demostrar que las ideas bajo las cuales se apoyaban dichos algoritmos podían llevar a conseguir exitosos resultados.

En general, cuando se afronta un problema utilizando técnicas meta heurísticas, como en el caso de los algoritmos de optimización basados en colonias de hormigas, es importante la aplicación del algoritmo a un amplio número de problemas con el objetivo de comparar los resultados obtenidos con los de otras técnicas que ya se encuentran actualmente disponibles para resolver dichos problemas. Además de ello, es necesario un estudio más profundo con el fin de demostrar si es posible encontrar una solución óptima al problema, en otras palabras, es necesario probar la convergencia del algoritmo usado. En éste aspecto, Gutjahr presentó las primeras pruebas de convergencia de un algoritmo de optimización basado en colonias de hormigas (Gutjahr., 2000) y (Gutjahr., 2002), llamado Graph-Based Ant System, el cual difiere ligeramente de los algoritmos de optimización basados en colonias de hormigas más populares usados en aplicaciones reales, por lo cual, dichos resultados obtenidos no pueden ser extrapolados directamente a otros algoritmos similares. Sin embargo, M. Dorigo y T. Stützle (Dorigo & Stützle., 2002) y (Dorigo & Stützle., 2004), mostraron resultados de convergencia para dos de los más importantes algoritmos de optimización basados en colonias de hormigas, Ant Colony System (AS) y MAX-MIN Ant System pero a pesar de ello, estas pruebas de convergencia no permiten predecir cuanto tiempo dichos algoritmos necesitan para encontrar una solución óptima. Más recientemente, Gutjahr (Gutjahr., 2006) presentó una estructura analítica con el fin de poder predecir de manera teórica la velocidad de convergencia de algoritmos de optimización basados en colonias de hormigas específicos.

La aplicación hoy en día de los diferentes algoritmos de optimización basados en colonias de hormigas se realiza en un amplio rango de problemas de optimización discretos. A pesar del hecho de que las aplicaciones se han incrementado enormemente en los últimos años, el uso principal de ésta serie de algoritmos es resolver problemas considerados inmersos en el campo de los problemas NP-duros, es decir, aquellos problemas para los cuales los mejores algoritmos conocidos para solucionarlos que garantizan encontrar una solución óptima toman tiempo exponencial. En éstos casos, los algoritmos ACO pueden son realmente útiles a fin de conseguir encontrar soluciones de alta calidad en un tiempo aceptable.

Como se ha comentado anteriormente, un requisito básico a la hora de implementar nuevas técnicas meta

heurísticas, es la aplicación de las mismas a un amplio rango de problemas a fin de compararla con las técnicas ya existentes. En éste aspecto, los algoritmos ACO han sido probados en más de cien problemas considerados NP-duros como problemas de ruteado en la distribución de bienes, problemas de asignación, en los cuales un conjunto de bienes (objetos, actividades, etc.) tienen que ser asignados a un número dado de recursos (localizaciones, agentes, etc.) sujetos a una serie de restricciones, o problemas de organización de tareas, los cuales, en el más amplio sentido, se tratan del reparto de una serie de recursos escasos a una serie de tareas a lo largo del tiempo. Es importante señalar que en la gran mayoría de las aplicaciones nombradas anteriormente, los algoritmos de optimización basados en colonias de hormigas que mejores resultados mostraron son aquellos que hacían un uso intensivo de la fase de búsqueda local que tiene lugar de forma opcional en el algoritmo meta heurístico basado en colonias de hormigas ya explicado anteriormente. Una vez estudiados los resultados proporcionados por los algoritmos meta heurísticos ACO en dichos problemas (NP-duros), se ha observado como en muchos de los casos, como problemas de ordenación, problemas de organización de horarios en la apertura de nuevas tiendas o muchas variantes de problemas de ruteo de vehículos, dichos algoritmos no solo proporcionan resultados similares a aquellos previos algoritmos utilizados para resolver dichos problemas sino que en general proporcionan incluso mejores resultados que éstos. Debido al éxito de dichos algoritmos en problemas académicos, numerosas compañías e investigadores están aplicando dichos algoritmos a aplicaciones reales, entre las cuales destaca AntRoute, herramienta desarrollada por la compañía AntOptima, para conseguir optimizar el ruteo de cientos de vehículos de compañías como Migros, la mayor cadena Suiza de supermercados o Barilla, la mayor compañía productora de pasta en Italia. En la siguiente tabla, 'Tabla 2: Lista no exhaustiva de las aplicaciones de los algoritmos basados en colonias de hormigas', se muestran algunas de las aplicaciones más notorias de los algoritmos de optimización basados en colonias de hormigas.

Tipo de problema	*Nombre del problema*	*Autores*	*Año*
Ruteo	Traveling salesman	Dorigo y otros.	1991, 1996
		Dorigo & Gambardella	1997
		Stützle & Hoss	1997, 2000
	Vehicle routing	Gambardella y otros.	1999
		Reimann y otros.	2004
	Sequential ordering	Gambardella y Dorigo	2000
Asignación	Quadratic assignment	Stützle & Hoss	2000
		Maniezzo	1999
	Course timetabling	Socha y otros.	2002,2003
	Graph coloring	Costa & Hertz	1997
Organización temporal	Project scheduling	Merkle y otros.	2002
	Total weighted tardiness	Den Besten y otros.	2000
		Merkle & Middendorf	2000
	Open Shop	Blum	2005
Subconjuntos	Set covering	Lessing y otros.	2004
	l-cardinality tres	Blum & Blesa	2005
	Multiple knapsack	Leguizamón & Michalewicz	1999
	Maximum clique	Fenet & Solnon	2003
Otros	Constraint satisfaction	Solnon	2000,2002
	Classification rules	Parpinelli y otros.	2002
		Martens y otros.	2006
	Bayesian networks	Campos, Fernández-Luna	2002
	Protein folding	Shmygelska & Hoos	2005
	Docking	Korb y otros.	2006

Tabla 2: Lista no exhaustiva de las aplicaciones de los algoritmos basados en colonias de hormigas

Para terminar con éste apartado, es necesario hacer una referencia a la dificultad que conlleva el hecho de determinar si un algoritmo pertenece o no al conjunto de algoritmos de optimización basados en colonias de hormigas. Ésto se debe principalmente a la enorme cantidad de definiciones que éstos han recibido por partes de diversos autores. La diferencia más común para determinar si un algoritmo pertenece o no a dicha familia, de acuerdo con varios autores, es precisamente el carácter constructivo que los mismos. En problemas combinatorios, utilizando dichos algoritmos, es posible que la mejor solución sea encontrada a pesar de que ninguna hormiga haya recorrido exactamente dicha solución, es decir, la mejor solución puede ser formada por el conjunto de las mejores soluciones parciales. Como ejemplo, en el caso del problema del vendedor viajante, no es necesario que una hormiga recorriera la mejor ruta ya que dicha ruta puede ser formada a partir de los ejes más fuertes de las mejores soluciones, como se puede observar en la Figura 2.

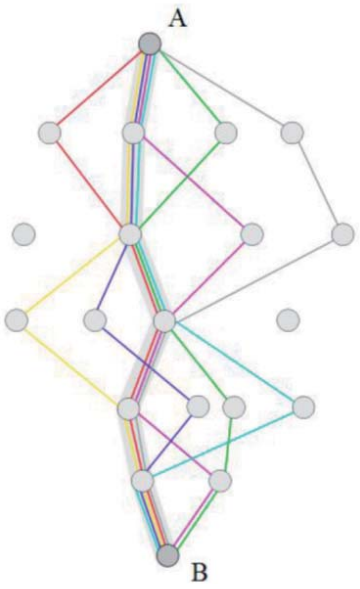

Figura 2: Recorrido más corto entre A y B

A pesar de que una vez dada ésta definición parece ser sencillo clasificar si un algoritmo está o no está dentro de los algoritmos de optimización basados en colonias de hormigas, en problemas con variables reales en los cuales no existe una estructura de 'vecinos' como en el problema del vendedor viajante, dicha clasificación puede resultar complicada.

El comportamiento social de conjuntos de insectos es actualmente una fuente de inspiración para diferentes investigadores. La amplia variedad de algoritmos, ya sean de optimización o no, que plantan sus bases en la propia auto-organización de sistemas biológicos ha llevado a la definición del concepto de métodos de inteligencia colectiva (Swarm Intelligence methods), en los cuales, obviamente, los algoritmos de optimización basados en colonias de hormigas pueden ser colocados. Algunos de los algoritmos más famosos que pueden ser considerados dentro de los métodos de inteligencia colectiva son los algoritmos genéticos (Genetic algorithms), los algoritmos de recocido simulado (Simulated annealing) o los algoritmos de búsqueda tabú (Tabu search) entre otros.

6.3. DESARROLLO DEL ALGORITMO

Durante el proceso de desarrollo de un algoritmo que resultara eficaz y útil para las diferentes instituciones u organismos interesados en solucionar el problema de ruteo de las máquinas quitanieves, en especial cuando éstas se enfrentan a grandes redes viarias, se han ido realizando diferentes modificaciones en el mismo con el fin de analizar cada una de las soluciones obtenidas y, de ésta forma, considerar si éstas modificaciones actúan positivamente o negativamente en el resultado final.

Por tanto, se puede considerar el proceso de trabajo como práctico y aplicado, ya que, el desarrollo del algoritmo ha estado guiado en todo momento por los resultados ofrecidos en las distintas modificaciones del mismo. De ésta forma, todas y cada una de las decisiones tomadas es justificada en base a resultados reales.

6.3.1. BASE DE TRABAJO DEL ALGORITMO

Con el propósito de que las pruebas realizadas en las diferentes modificaciones del algoritmo fueran aplicadas en un entorno lo más próximo a la realidad, y, por tanto, que la justificación de adoptar o no dichas modificaciones como finales estuvieran totalmente justificadas, se consideró necesario realizar un grafo que simulara la verdadera entidad real del problema.

Puesto que el programa de intercambio se ha llevado a cabo en la Universidad de Rhode Island, y, debido al enorme interés mostrado por la institución para abordar dicho problema, debido a las extremas condiciones climáticas que ésta sufre en cada temporada invernal, teniendo en múltiples ocasiones que suspender clases y actividades debido al enorme calado del problema, se consideró apropiado que el grafo a realizar, en el cual se aplicaran las diferentes pruebas del algoritmo, fuera la propia universidad. De ésta forma, generando dicho grafo a imagen del propio campus de Kingston de la Universidad, con 201 nodos simulando los cruces de calles y más de 260 ejes simulando dichas calles, se consigue obtener un grafo que aborde la enorme entidad del problema.

Para poder llevar a la práctica dicha idea y que la localización de los nodos (cruces de calles) tuviera un sentido real, se utilizó Google Earth con el fin de localizar geográficamente dichos nodos. Para ello, se establecieron etiquetas con Google maps indicando cuales eran dichos puntos y, a partir de ello, puesto que dicho programa proporciona las coordenadas UTM de cada uno de ellos, los nodos del grafo fueron implementados en el algoritmo poniendo como localizador de los mismos dicha información geográfica. En la Figura 3 se puede ver cómo se etiquetaron todos los nodos del campus gracias a las aplicaciones de Google y, posteriormente, en la Figura 4, como se realizó el grafo del campus basándose en la información obtenida.

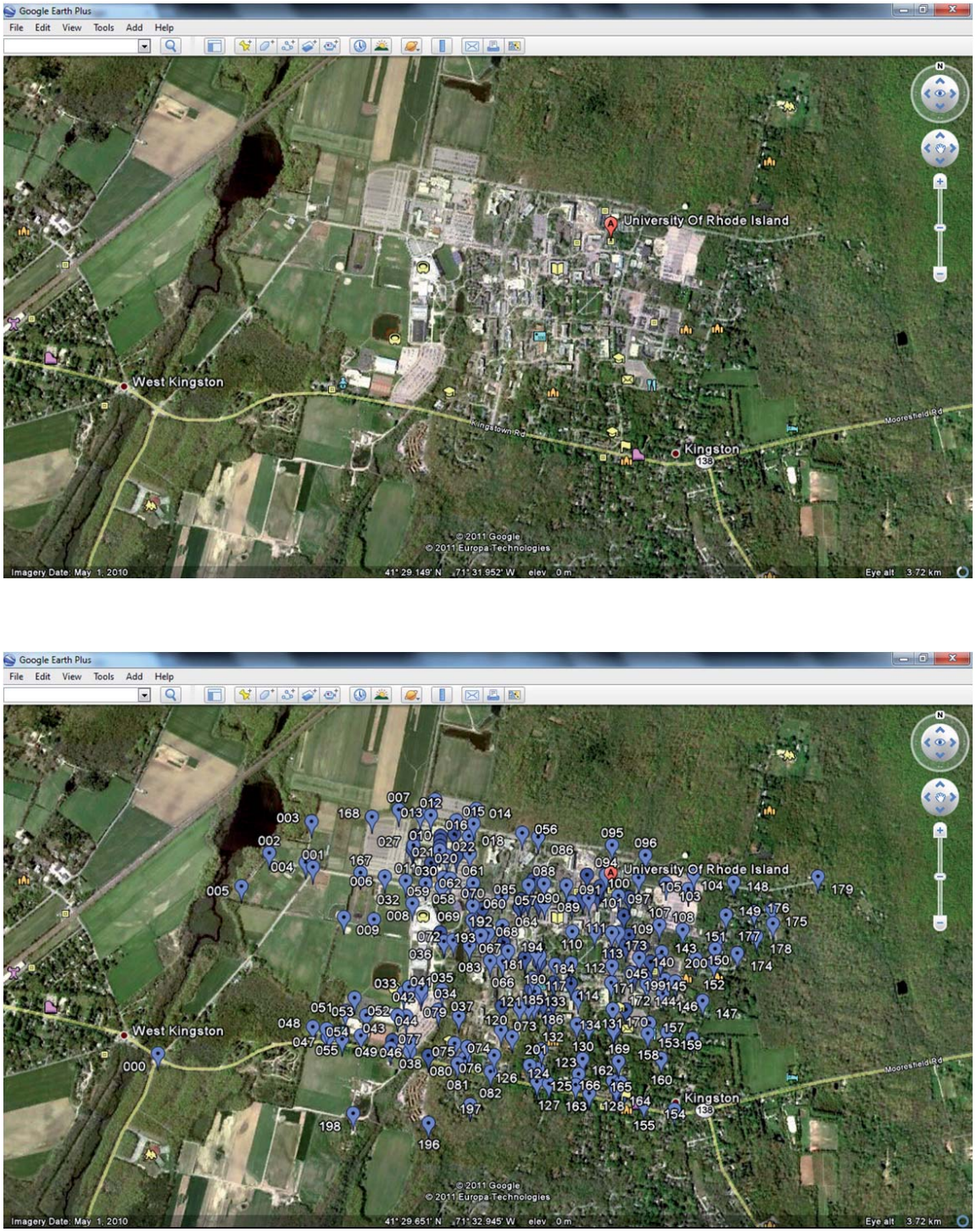

Figura 3: Establecimiento de etiquetas con Google Earth

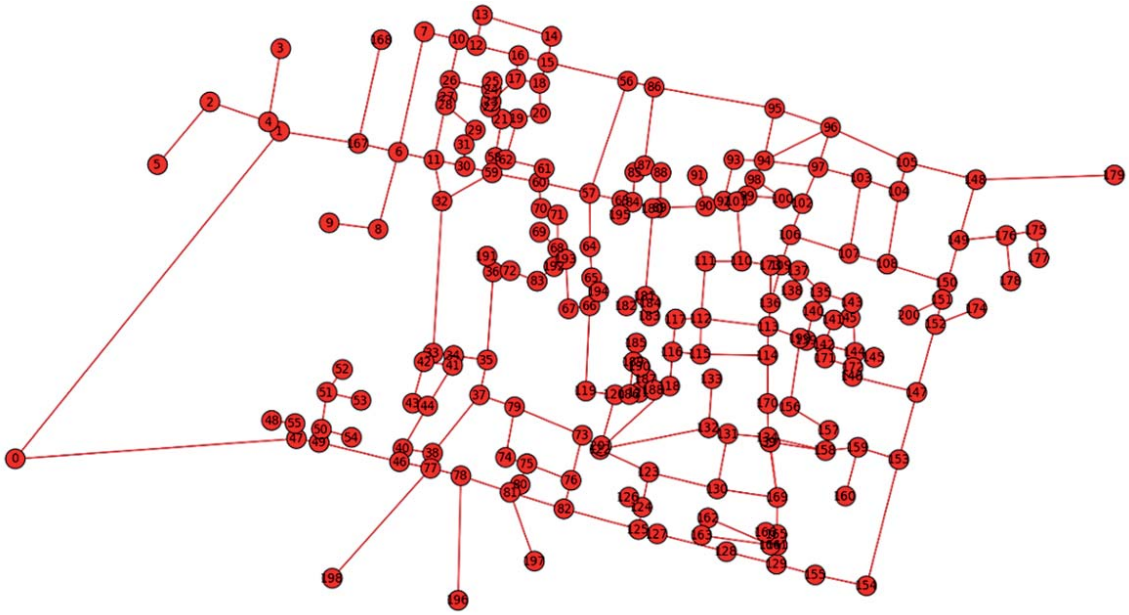

Figura 4: Grafo realizado de la universidad

Sin embargo, debido a la enorme carga computacional que estos algoritmos suponen, se consideró necesario, a fin de acelerar las pruebas, utilizar un grafo de tan solo 9 nodos y 13 ejes en alguna de las modificaciones del algoritmo como paso previo a la aplicación de éstos en el grafo completo de la universidad. De ésta forma, además de acelerar el proceso, se consigue hacer un seguimiento mucho más detallado del proceso de decisión llevado a cabo por el algoritmo. Dicho grafo se puede apreciar en la Figura 5.

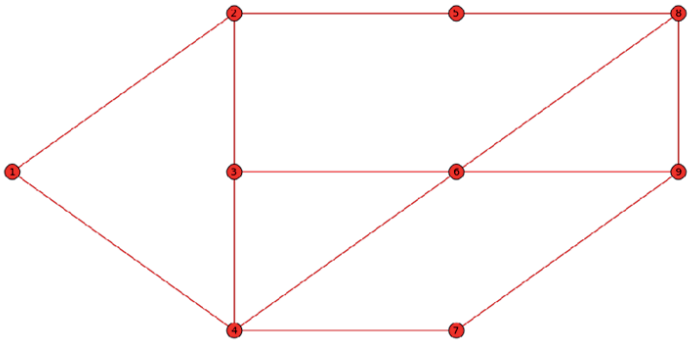

Figura 5: Grafo utilizado de prueba

6.3.2. BASES DE DESARROLLO DEL ALGORITMO

Como ya se ha comentado en el apartado correspondiente a los algoritmos de optimización basados en colonias de hormigas, generalmente, para poder decir que un algoritmo se encuentra inmerso en dicha familia, es fundamental que éste tenga un carácter constructivo, es decir, en el caso de los problemas combinatorios, que la mejor solución encontrada pueda ser formada por el conjunto de las mejores soluciones parciales sin que, en realidad, ninguna de las soluciones dadas en las diferentes iteraciones sea en sí misma la mejor solución.

Por éste motivo, es necesario notar que el algoritmo desarrollado no se puede considerar inmerso en dicha familia ya que, en este caso, el resultado final siempre será la solución completa obtenida en la mejor iteración, sin que éste resultado final pueda ser formado por distintas soluciones parciales encontradas en diferentes iteraciones.

A pesar de ello, las bases sobre las que se asienta el algoritmo desarrollado son los principios fundamentales de la familia de algoritmos de optimización basados en colonias de hormigas. El algoritmo, el cual se puede considerar inmerso en el campo de la meta heurística, adopta diferentes ideas de los métodos más famosos considerados dentro de la familia de 'inteligencia colectiva', aunque como ya se ha comentado, con un enorme hincapié en los algoritmos basados en colonias de hormigas. Además de ello, con el fin de mejorar los resultados obtenidos, varias de las modificaciones de dicho algoritmo se han realizado en base a una experiencia práctica, es decir, mediante el uso de la lógica y el apoyo de los resultados obtenidos en el grafo completo de la universidad de Rhode Island.

6.3.3. CONSTRUCCIÓN DEL ALGORITMO

Tal y como ya se puntualizó en la introducción del presente proyecto, el objetivo principal del mismo es la construcción de un programa que permita establecer la ruta más adecuada a seguir por los vehículos encargados de realizar las operaciones de mantenimiento invernal de la red viaria. Para ello, en un principio se resuelve el problema conocido como 'Chinese Postman Problem', en el cual, todos los ejes de un determinado grafo, el cual representa el conjunto viario en cuestión, deben ser recorridos al menos una vez. Puesto que los vehículos deben de comenzar y terminar su trabajo en un punto determinado del grafo, el cual representa tanto los depósitos de material, de nieve o incluso de vehículos, tal y como se señaló en el apartado '3.2. Gestión de los problemas de organización operacionales', una vez solucionado el ya nombrado 'Chinese Postman Problem' se fuerza a los vehículos a que retornen a su punto inicial de partida, con el propósito de simular exactamente el procedimiento que estos siguen durante las operaciones de mantenimiento invernal.

A lo largo del desarrollo del algoritmo, una vez se considera que el 'Chinese Postman Problem' ya ha sido resuelto de una manera satisfactoria, se procede a resolver otro problema de similar entidad llamado 'Rural Postman Problem', en el cual, ya no todos los ejes de un grafo deben ser recorridos, sino que tan solo algunos de ellos deben serlos. Dicha modificación de los problemas se lleva a cabo para simular de una manera más real dichas operaciones de mantenimiento, ya que, por lo general, no siempre es necesario que todas las carreteras de un conjunto viario sean limpiadas, sino que solo un subconjunto de ellas lo sean, debido a diferentes clasificaciones jerárquicas, tal y como ya se explicó en el apartado '3.2.1. Nivel de servicio'.

A DÍA 18_1_2011:

Este primer programa desarrollado trata simplemente de conseguir obtener unos valores aproximados de la longitud que pueden tener los recorridos en los grafos, y de esta forma, poder analizar la conveniencia del uso de alguna técnica específica que mejore el proceso.

En primer lugar, se asigna a cada eje el parámetro 'weight', con el objetivo de memorizar si dicho eje ha sido ya recorrido o no con el fin de determinar cuando el programa debe finalizar el proceso. Una vez un eje pasa de no recorrido a recorrido, el valor del parámetro 'weight' se actualiza de 0.0 a 10.0. Sin embargo, cuando un eje que ya ha sido recorrido vuelve a serlo, el valor del 'weight' no se vuelve a actualizar, sino que éste se queda en 10.0.

El proceso de selección de los ejes se ejecuta seleccionando prioritariamente aquellos ejes vecinos que todavía no han sido recorridos, es decir, de entre aquellos ejes adyacentes con valor 'weight = 0.0' se realiza una selección aleatoria de cuál de ellos debe ser escogido como el siguiente. En el caso de que todos los ejes adyacentes a un nodo hayan sido recorridos, es decir, todos tengan 'weight=10.0' la selección se realiza de forma totalmente aleatoria entre todos ellos.

El programa continúa con el proceso hasta que todos y cada uno de los ejes tienen valor 10.0, es decir, todos ellos han sido recorridos. Una vez esto ocurre, puesto que en el proceso de ruteo de las quitanieves es necesario que éstas vuelvan al depósito, se utiliza un algoritmo llamado 'shortest path' proporcionado por Networkx, el cual nos proporciona el recorrido más corto, es decir, con menos calles, desde el nudo en que el programa ha finalizado hasta el depósito.

CONCLUSIONES:

Tras la aplicación del mismo en el grafo pequeño de 9 nodos, se demostró que la distancia de los recorridos era bastante variable, habiendo desde recorridos de 17 ejes hasta recorridos de 32 ejes, por lo que los resultados no son útiles, pero éstos nos dan una idea de la dificultad de realizar un programa efectivo para éste tipo de problemas y planta las bases para comparar los resultados de los siguientes programas realizados. Además, tras aplicar dicho programa en el grafo de la universidad se observa como, al tratarse de un programa de selección totalmente aleatorio y dada la magnitud del grafo, el tiempo que el programa necesita para lograr una solución, así como la calidad de la misma son totalmente aleatorios.

A DÍA 20_1_2011:

Tras el análisis detallado del procedimiento de trabajo del programa anterior, se ha considerado como otra alternativa el almacenar la información referente al número de veces que un eje es atravesado y utilizarla en el proceso de selección del siguiente eje a recorrer. Para ello, se establece en todos los ejes un valor de 'weight=1.0' y, cada vez que un eje es recorrido, se suma un valor determinado a dicho parámetro, el cual se ha ido cambiando en las diferentes pruebas realizadas.

A diferencia del programa anterior, en el cual la selección de ejes se realizaba de forma totalmente aleatoria, en éste caso dicha selección es función de un parámetro p probabilístico dependiente del número de veces que un eje ha sido recorrido. De ésta forma, no se tiene en cuenta si un eje ha sido o no recorrido previamente, como se hacía en el programa anterior, sino que se valora el número de veces que éste ha sido recorrido.

Dicha probabilidad p para que un eje i sea seleccionado se ha establecido de la siguiente forma:

$$p_i = \frac{1}{n-1} * \frac{S - w_i}{S}$$

$$S = w_1 + w_2 + w_3 + \cdots + w_n$$

Ecuación 3: Probabilidad de selección de un eje

donde S es la suma del parámetro 'weight' de todos los ejes anexos a un nodo y n es el número de ejes anexos.

Una vez todos los ejes adyacentes tienen un valor asociado de probabilidad, el cual es almacenado en un nuevo parámetro llamado 'probability', y puesto que $p_1 + p_2 + p_3 + \cdots + p_n = 1$, se establece una probabilidad total a cada uno de ellos siendo:

$$p_{1\,total} = 0 + p_1$$

$$p_{2\,total} = p_{1\,total} + p_2$$

$$p_{3\,total} = p_{2\,total} + p_3$$

$$p_{n\,total} = p_{(n-1)total} + p_{(n-1)}$$

Ecuación 4: Probabilidades totales

Una vez esto es realizado se genera un número aleatorio $r = [0,1)$, y si $r < p_{1\,total}$ el eje $i = 1$ es seleccionado, si $p_{1\,total} < r > p_{2\,total}$, el eje $i = 2$ es seleccionado y así sucesivamente. De ésta forma, contra más veces un eje haya sido recorrido, menor es la probabilidad de que éste sea elegido de nuevo.

Con éste mecanismo de selección de recorrido, el cual tiene en cuenta cuantas veces un eje ha sido visitado anteriormente, se consigue evitar parcialmente la formación de bucles en la solución final ya que cuando un eje ha sido visitado varias veces la probabilidad de que éste vuelva a ser escogido se reduce enormemente mejorando de ésta forma las soluciones obtenidas, tal y como se verá a continuación en los diferentes programas. Éste procedimiento de eliminación de bucles, si bien no es exactamente un procedimiento ''a posteriori'' de mejora de las soluciones obtenidas, ya que se va realizando al mismo tiempo que se construye la solución, se puede considerar inmerso en la fase de ''Aplicar búsqueda local'' la cual, como ya se nombró en el capítulo correspondiente, aun siendo opcional, se aplica en la gran mayoría de los algoritmos de optimización basados en colonias de hormigas con el fin de mejorar las soluciones obtenidas. Por tanto, el algoritmo usado en éste programa y en los siguientes posee dos fases principales en lugar de las tres habituales en los algoritmos de optimización basados en colonias de hormigas, ya que la primera y segunda fase de éstos, ''Construcción de soluciones'' y ''Aplicar búsqueda local'' se encuentran unidos en una misma fase en el programa desarrollado manteniendo común la tercera fase del algoritmo, ''Actualizar feromonas''.

Finalmente, una vez todos los ejes han sido recorridos, es decir, el parámetro 'weight' de cada uno de los ejes es mayor que 1.0, se utiliza de nuevo el algoritmo 'shortest path' con el fin de conocer el camino más corto al depósito desde el último nodo visitado.

CONCLUSIONES:

El programa, a pesar de que una vez comparado con el anterior en el grafo de 9 nodos genera peores soluciones, se debe considerar que a la larga, en grafos mucho más grandes, dicho procedimiento de selección del eje en función del número de veces que éste es recorrido puede llegar a ser realmente útil.

Por tanto, con el propósito de analizar cuál es el valor óptimo a sumar al parámetro 'weight' cada vez que un eje es atravesado y así optimizar el uso de ésta técnica para posibles posteriores usos de la misma, se han realizado varias pruebas con distintos valores. Los resultados son mostrados en el Gráfico 1 para un total de 1000 iteraciones para cada uno de los valores.

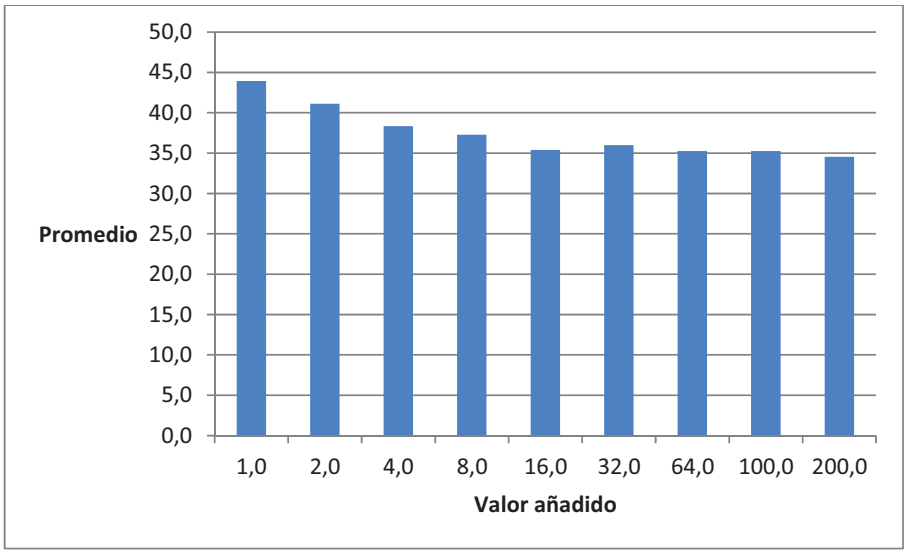

Gráfico 1: Promedio establecido con 1000 iteraciones para los diferentes valores añadidos.

Analizando éstos resultados obtenidos se puede observar cómo una vez el valor añadido se encuentra por encima de 8.0, los cambios en los resultados ya no son significativos, por lo que se puede considerar para siguientes versiones del programa que un valor alrededor de 10.0 es más que suficiente.

A DÍA 23_1_2011:

En esta modificación del programa, se ha analizado si es más favorable realizar el incremento del parámetro 'weight', a fin de marcar el número de veces que un eje es recorrido, mediante el mecanismo usado en el programa anterior, es decir, mediante un incremento fijo del mismo de manera sumatoria o si es más conveniente realizarlo de forma multiplicativa, es decir, multiplicando el valor de dicho parámetro por una cantidad determinada cada vez que el eje es atravesado.

El hecho de que el parámetro 'weight' se incremente de forma multiplicativa hace que el programa desarrollado todavía penalice más el hecho de que un eje haya sido recorrido previamente por lo que además, la condición de eliminación de bucles todavía se hace más restrictiva. Por tanto, es necesario comprobar si éste nuevo mecanismo actúa de manera positiva o negativa en los resultados finales del programa. El resto del programa es totalmente idéntico al anterior.

CONCLUSIONES:

A fin de analizar los resultados, se han realizado 1000 iteraciones del programa sobre el grafo de 9 nodos para diferentes valores multiplicativos del parámetro 'weight'. Dichos resultados se pueden observar en el Gráfico 2.

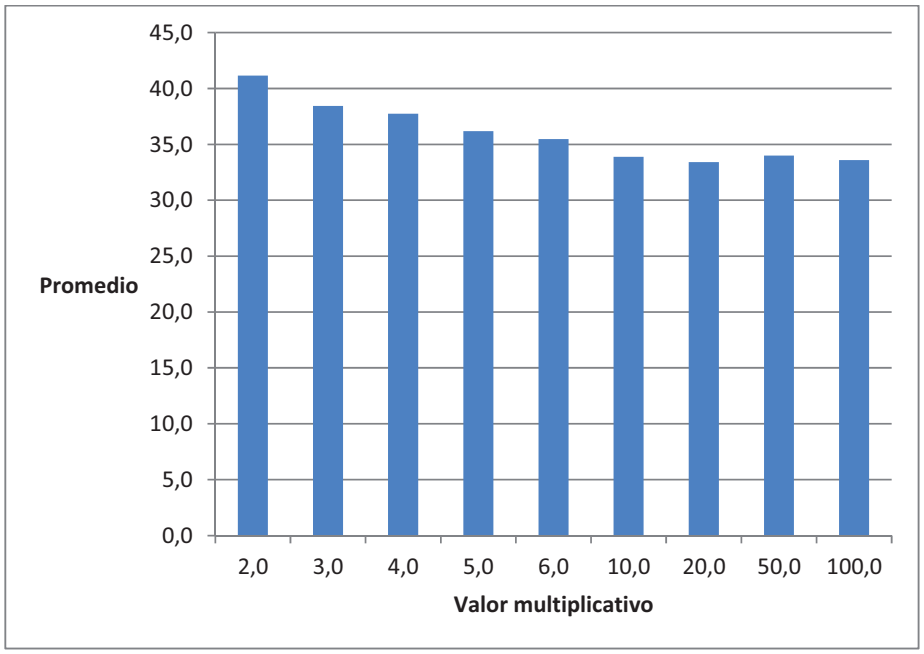

Gráfico 2: Promedio establecido con 1000 iteraciones para los diferentes valores multiplicativos.

Una vez comparados éstos resultados con los proporcionados con el programa anterior, los cuales se pueden observar en el Gráfico 1, en el cual el valor del parámetro 'weight' se iba incrementando de manera sumatoria, se puede ver cómo, en general, los resultados de ésta versión del programa son ligeramente mejores a los obtenidos en el caso del programa anterior, por ello, se considera como una medida de mejora el realizar la actualización del parámetro 'weight' de manera multiplicativa.

Sin embargo, analizando detalladamente éstos resultados, se puede observar cómo, a partir de valores multiplicativos de 10.0, los resultados obtenidos ya no mejoran significativamente. Por ello, se puede considerar dicho valor como un límite máximo para los próximos programas desarrollados que utilicen ésta técnica.

A DÍA 26_1_2011:

Como se estudió anteriormente en el capítulo '6.2. Algoritmos de optimización basados en colonias de hormigas (ACO)', una de las principales características de ésta familia de algoritmos es la existencia de la llamada retroalimentación negativa, por la cual, el valor de feromonas presente en un eje va desapareciendo de forma progresiva. Con éste mecanismo se consigue que la información que ha sido depositada en cada eje vaya desapareciendo con el paso del tiempo y de ésta forma, en aquellos recorridos con peores resultados, es decir, los más largos, dichos valores serán menores que aquellos con recorridos más cortos, mostrando así la mayor conveniencia de éstos últimos.

Con el fin de utilizar dicho concepto de una manera aproximada en el programa desarrollado, se ha introducido un factor de reducción del parámetro 'weight' de cada eje, el cual, como recordatorio, se incrementa de una forma multiplicativa cada vez que éste es atravesado en una iteración, reseteándose dicho valor a 1.0 cuando la iteración ha concluido. De ésta manera se pretende que el hecho de que un eje que haya sido recorrido hace bastante tiempo, poco a poco vaya eliminando sus marcas indicativas de que éste ha sido seleccionado anteriormente y de ésta manera que la penalización por ello no sea tan fuerte. Éste factor de

reducción tiene las siguientes características:

- Es directamente multiplicativo con el parámetro 'weight' de cada eje, por lo que contra menor sea dicho parámetro de reducción, menor será la penalización de que un eje haya sido visitado anteriormente.
- Éste actúa cada vez que un eje es recorrido en una determinada iteración.
- Actúa en todos los ejes menos en aquel que está siendo recorrido.
- Solo se aplica dicha reducción cuando el valor final del 'weight' tras la reducción se encuentre por encima del valor 2.0. Esta característica hace que se impida borrar la marca de que un eje ya ha sido recorrido al menos una vez, es decir, que el 'weight'≠1.0. La importancia de evitar esto se debe a que una iteración se considera finalizada cuando el 'weight' de todos los ejes es distinta de 1.0 y por tanto, en caso de no utilizar ésta regla, la iteración podría entrar en un bucle infinito.

CONCLUSIONES:

Para analizar el comportamiento de dicha modificación se ha aplicado el programa al grafo de 9 nodos para diferentes valores tanto del factor multiplicativo 'weight' como del factor reductor. Los valores del factor multiplicativo se mueven entre 2.0 y 10.0 ya que, como se analizó en programas anteriores, valores superiores a 10 no traían mejoras significativas. En cuanto al factor de reducción se ha ido variando desde valores que no penalizan tan apenas que un eje haya sido recorrido, 0,1 hasta valores que lo penalizan enormemente, 0,9.

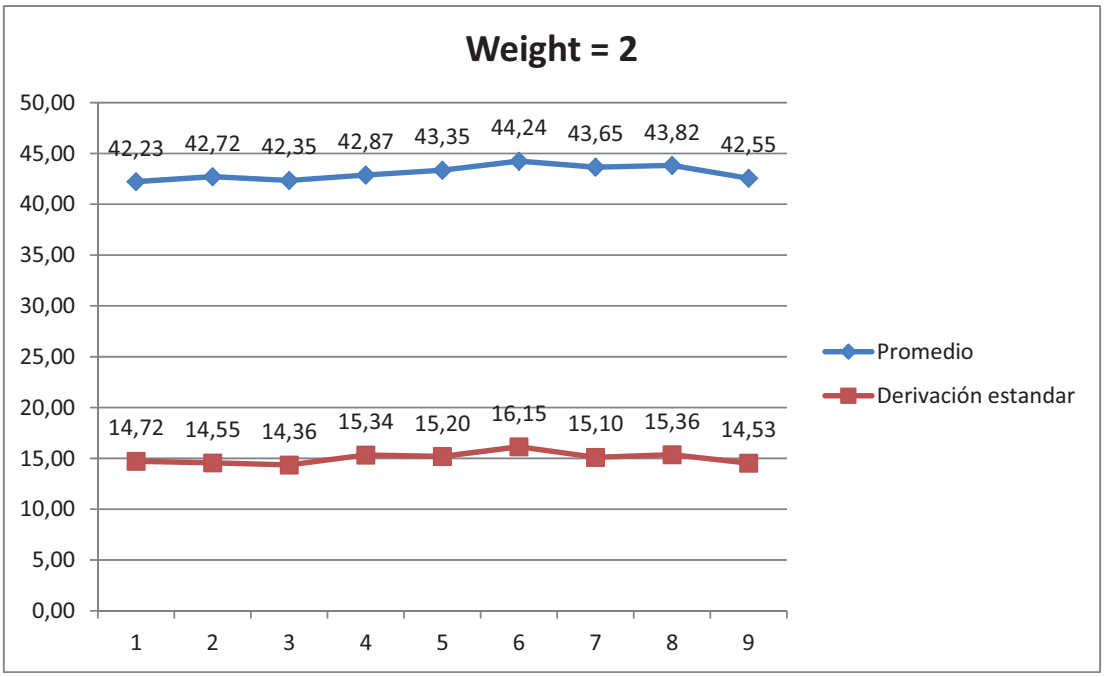

Gráfico 3: Valores obtenidos para valor multiplicativo de 2.0

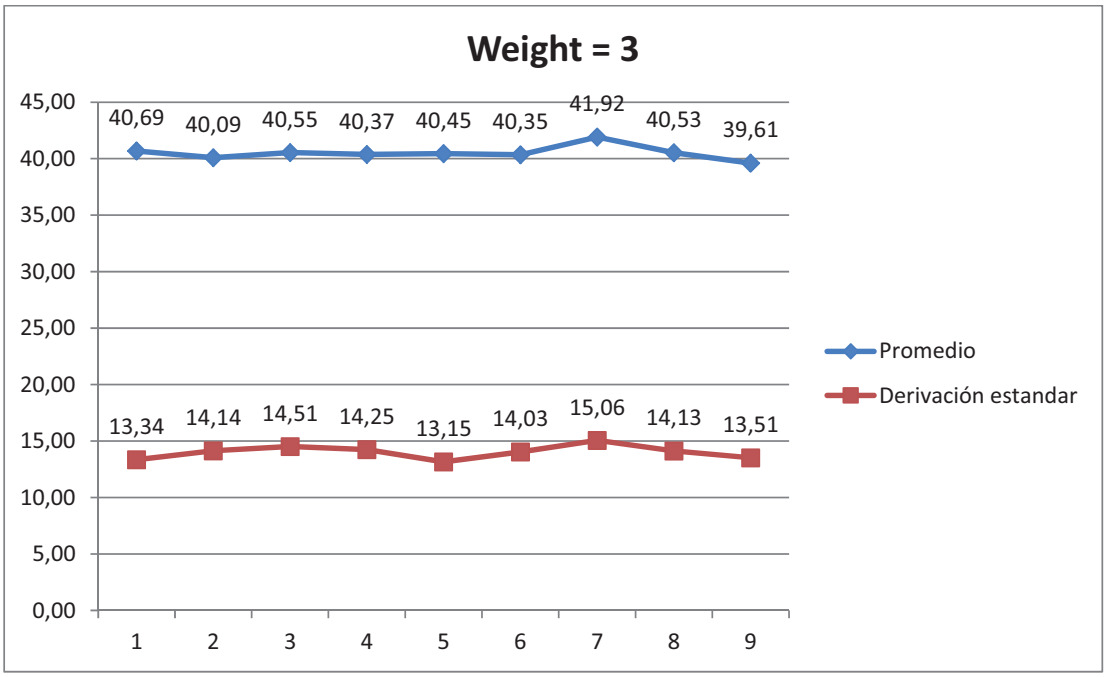

Gráfico 4: Valores obtenidos para valor multiplicativo de 3.0

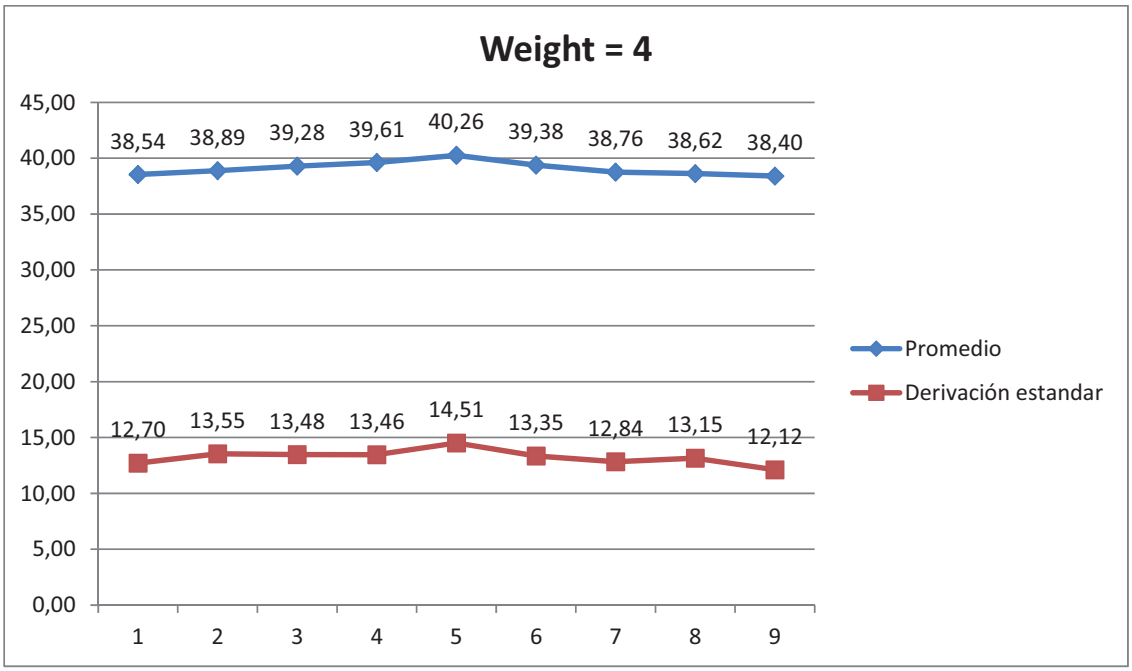

Gráfico 5: Valores obtenidos para valor multiplicativo de 4.0

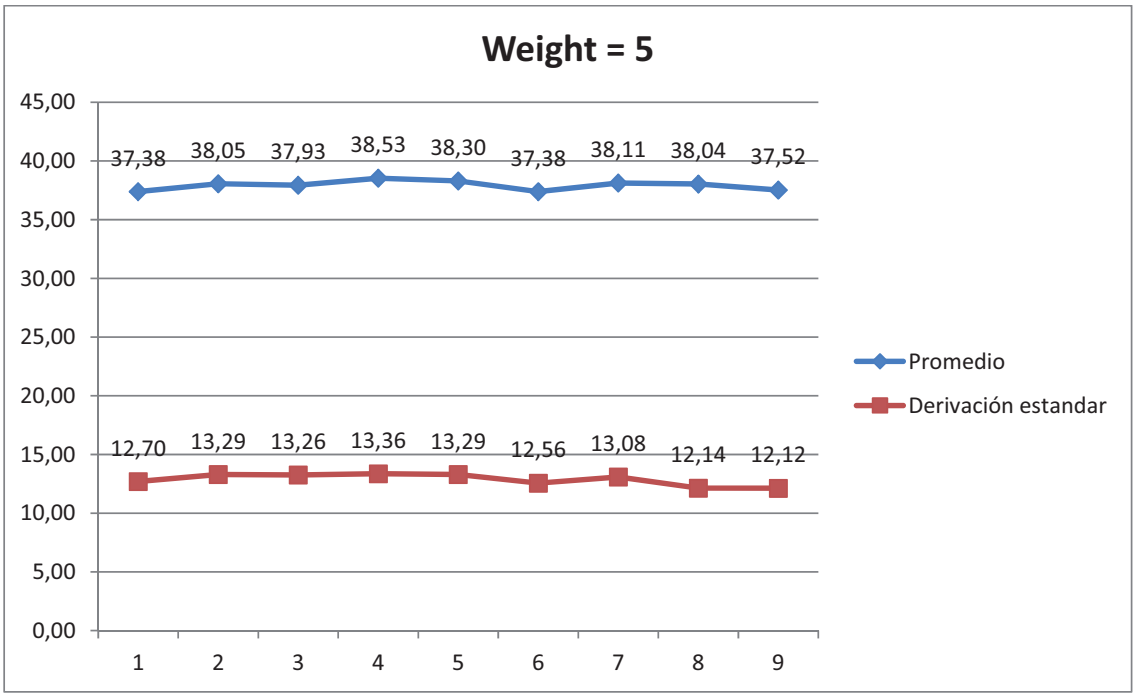

Gráfico 6: Valores obtenidos para valor multiplicativo de 5.0

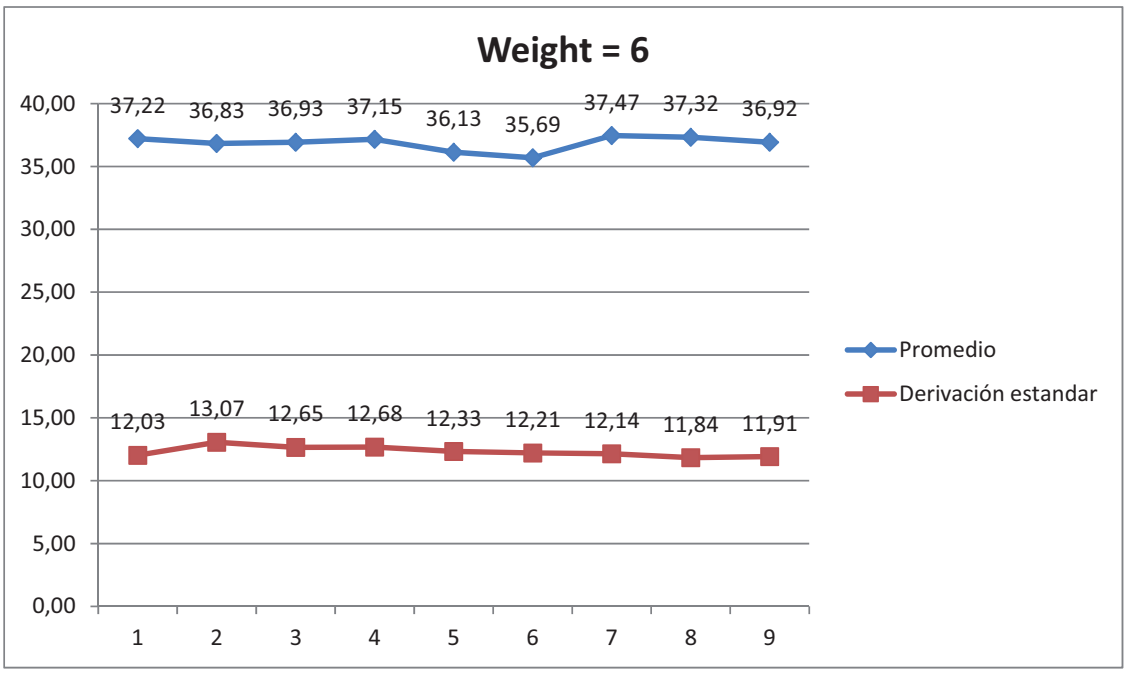

Gráfico 7: Valores obtenidos para valor multiplicativo de 6.0

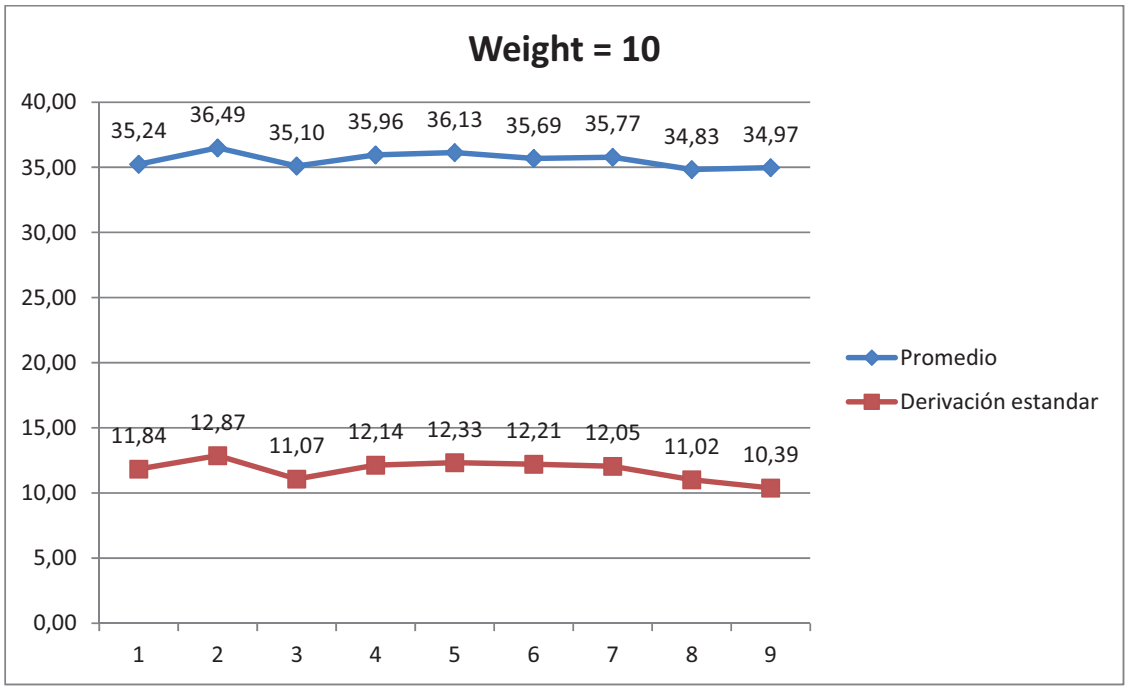

Gráfico 8: Valores obtenidos para valor multiplicativo de 10.0

Los valores observados en los gráficos anteriores, Gráfico 3, Gráfico 4, Gráfico 5, Gráfico 6, Gráfico 7 y Gráfico 8, muestran como tanto la derivación estándar como el promedio se incrementan a medida que se reduce el valor multiplicativo del 'weight' como ya se observó en pruebas anteriores realizadas. Por otra parte, en cuanto al factor de reducción se refiere, se puede observar como éste no actúa de una manera significativa en los resultados obtenidos, ya que los valores de dicho factor actúan de formas muy dispares en referencia a los diferentes factores multiplicativos.

A DÍA 2_2_2011:

A fin de realizar una comparativa óptima y conseguir obtener conclusiones finales en cuanto a las distintas versiones del programa que hasta el momento se dispone, es necesario aplicar dichos programas a un grafo de entidades reales como es el grafo de la Universidad de Rhode Island.

Los resultados para las tres modificaciones más importantes del programa hasta el momento se muestran a continuación. En cada uno de ellos se han realizado 100 iteraciones y se han obtenido el valor medio, el valor máximo, el mínimo y la derivación estándar. Éste último valor resulta de vital importancia ya que para conocer con detalle un conjunto de datos, no basta con conocer las medidas de tendencia central, sino que necesitamos conocer también la desviación que representan los datos en su distribución respecto de la media aritmética de dicha distribución, con objeto de tener una visión de los mismos más acorde con la realidad al momento de describirlos e interpretarlos para la toma de decisiones.

Programa a día 1_18_2011:

Programa 1_18_2011	
Promedio	3160.61
Derivación estándar	1365.22
Mínimo	1242
Máximo	8493

Tabla 3: Resultados para el programa 1_18_2011

Tras analizar los resultados obtenidos, se puede observar como el valor promedio de las 100 iteraciones es un valor bastante aceptable y, sobre todo, el valor mínimo es realmente bueno. A pesar de que el objetivo final del programa es proporcionar un solo recorrido como solución óptima, es necesario nombrar que dicho valor mínimo encontrado se trata simplemente de un valor encontrado de forma casual y que, en otras iteraciones podría llegar a no darse nunca un valor de tan buena calidad. Esto se debe a que el procedimiento de recorrido del grafo se realiza de una manera totalmente aleatoria cuando los ejes adyacentes a un nodo ya han sido visitados. A consecuencia del carácter aleatorio del programa, también se puede observar como el valor de la derivación estándar es realmente grande y, por tanto inaceptable. Por estos motivos, dicho programa queda descartado de posibles modificaciones o mejoras futuras.

Programa a día 1_23_2011:

Programa 1_23_2011						
Valor multiplicativo	2	3	4	5	6	10
Promedio	3013.4	2696.9	2566	2491.5	2558.9	2412.8
Derivación estándar	741.4	644.3	623.8	510.5	675.1	581.3
Mínimo	1708	1645	1409	1447	1348	1442
Máximo	6126	4817	4223	3569	4616	4219

Tabla 4: Resultados para el programa 1_23_2011

Al igual que se observó en las pruebas realizadas en el grafo de 9 nodos al aplicar dicho programa, se puede ver como los mejores resultados son los obtenidos con un valor multiplicativo de 10.0. Los resultados promedio son realmente buenos, llegando incluso a bajar de los 2500 ejes visitados. Sin embargo, el aspecto más importante a remarcar es el hecho de que se obtienen unos valores de derivación estándar realmente buenos, llegando incluso a mejorar en más de un 50% los valores obtenidos con el programa anterior. Éste hecho hace que, a pesar de que el valor mínimo obtenido no sea tan bueno como en el programa anterior, el procedimiento de trabajo muestra una fuerte consistencia y por tanto indica que éste sea el adecuado a seguir para futuras mejoras o modificaciones del programa.

Programa a día 1_25_2011:

Programa 1_25_2011									
Valor multiplicativo	2								
Factor reductor	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Promedio	4399.0	4611.0	4748.6	4718.2	5040.1	4452.9	5018.6	4679.5	4387.7
Derivación estándar	1539.1	1510.3	1621.1	1689.4	1746.1	1350.0	1974.7	1626.9	1404.1
Mínimo	2313	2423	2093	1870	2376	2158	2551	2556	1877
Máximo	9047	9509	11457	12039	11027	8753	16662	12732	9717

Tabla 5: Resultados para el programa 1_25_2011 con valor multiplicativo de 2.

Programa 1_25_2011									
Valor multiplicativo	3								
Factor reductor	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Promedio	4266.5	4338.3	4205.6	4143.4	4246.6	4250.0	4409.6	4182.5	4204.3
Derivación estándar	1523.0	1405.0	1621.1	1233.6	1479.1	1655.5	1852.1	1367.4	1383.7
Mínimo	2264	2317	2078	2282	1951	2063	1794	2111	2228
Máximo	11487	9431	12312	9264	10539	10447	11535	8576	11709

Tabla 6: Resultados para el programa 1_25_2011 con valor multiplicativo de 3.

Programa 1_25_2011									
Valor multiplicativo	4								
Factor reductor	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Promedio	4122.0	3831.5	4009.7	3813.5	3877.2	4191.1	3773.9	3970.0	3610.6
Derivación estándar	1356.0	1166.5	1364.2	1382.7	1162.3	1498.3	1263.9	1502.8	1171.7
Mínimo	1511	1566	1688	1876	1826	1879	1728	2004	1930
Máximo	8206	7294	9960	9434	8488	8844	9954	12002	7129

Tabla 7: Resultados para el programa 1_25_2011 con valor multiplicativo de 4.

Programa 1_25_2011									
Valor multiplicativo	10								
Factor reductor	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Promedio	3300.2	3278.6	3263.5	3386.3	3279.1	3574.0	3461.7	3230.4	3323.3
Derivación estándar	1102.0	1073.5	1344.0	1130.4	1073.4	1112.7	1112.4	1131.7	1153.8
Mínimo	1800	1779	1817	1715	1622	1995	1533	1475	1575
Máximo	6884	7764	10665	7223	7545	8169	7531	8422	9997

Tabla 8: Resultados para el programa 1_25_2011 con valor multiplicativo de 10.

Tras una observación detallada de los diferentes resultados obtenidos al aplicar el programa con distintos valores multiplicativos y factores reductores se puede observar como la media de los resultados va en decremento con el valor multiplicativo, cosa que era de esperar ya que, como ya se dijo anteriormente, el valor multiplicativo de 10.0 es el óptimo a utilizar. Por otra parte, se observa como el hecho de variar el factor reductor entre valores de 0.1 y 0.9, los resultados obtenidos varían enormemente, por lo que, al igual que se dedujo de la aplicación del programa al grafo de 9 nodos, dichos valores reductores no ofrecen una mejora sustancial de los resultados. Además de ello, como se puede observar, los valores de desviación estándar al usar dicho factor reductor son enormes para todos y cada uno de los valores multiplicativos por lo que se puede concluir que la metodología usada en ésta versión del programa no es la correcta para futuras modificaciones y, por tanto, ésta va a ser descartada.

A DÍA 2_8_2011:

Tras el análisis de los resultados anteriores al aplicar los diferentes programas al grafo de la Universidad de Rhode Island, se ha decidido continuar el desarrollo del programa a partir del realizado el día 1_23_2011 para un valor multiplicativo de 10.0 puesto que éste mostraba las mejores soluciones.

A fin de continuar con un desarrollo guiado por los conceptos básicos de la familia de algoritmos de optimización basados en colonias de hormigas, en ésta modificación del programa se añade un nuevo

parámetro a todos y cada uno de los ejes del grafo llamado 'pheromone'. La misión principal de éste parámetro es conseguir marcar la conveniencia de un recorrido por encima de otro a fin de que posteriores iteraciones mejoren su procedimiento de selección. Éste parámetro trata de simular de forma similar el concepto de depósito de feromonas estudiado en las colonias de hormigas cuando éstas encuentran un camino entre la fuente de comida y la colonia. De ésta forma, aquellos recorridos más cortos, deberán ser marcados más fuertemente que aquellos más largos con el propósito de marcar su conveniencia.

Para calcular los valores de 'pheromone' de cada eje y, a partir de ello, que las futuras iteraciones tengan la información necesaria para decidir que eje visitar, es necesario realizar una serie de recorridos a modo de inicialización. Una vez éste conjunto de recorridos, m ya haya sido realizado, será posible actualizar el parámetro 'pheromone' de los ejes.

Para ello, el conjunto de iteraciones de inicialización, va a ser realizado, como ya se nombró anteriormente, a partir del programa desarrollado el 1_23_2011 y usando un valor multiplicativo de 10.0. La decisión de realizar un conjunto de iteraciones de inicialización antes de actualizar el valor de 'pheromone' de los ejes en lugar de realizar dicha actualización tras cada iteración es debido a que, de ésta manera, la información inicial es mucho más completa y, además, el tiempo de ejecución del programa se reduce enormemente.

A partir del conjunto de recorridos m, obtenidos con las iteraciones de inicialización, se calcula en primer lugar cual es el recorrido más corto de todos ellos $\min(longitud_1^m)$. Una vez esto ha sido realizado, se calcula el valor de 'pheromone' que se debe añadir a cada uno de los ejes por los cuales una iteración ha pasado. Dicho valor es calculado como se muestra en la Ecuación 5 para un determinado recorrido X.

$$pheromone_X = e^{-4*\left(1-\left(\frac{longitud_X}{\min(longitud_1^m)}\right)^2\right)}$$

Ecuación 5: Actualización del parámetro 'pheromone'

Una vez dicho valor ha sido calculado, se procede a añadir dicho valor a todos los ejes por los cuales dicha iteración X ha pasado. Es necesario nombrar que, en caso de que un eje haya sido recorrido más de una vez en la misma iteración, el parámetro 'pheromone' que éste adquirirá será la suma del dicho parámetro $pheromone_X$ tantas veces como éste aparezca en la iteración X.

Tras realizar el mismo procedimiento para todas y cada una de las iteraciones de inicialización, m, los valores del parámetro 'pheromone' de cada uno de los ejes ya se encuentran en disposición de ser usados en las siguientes iteraciones que lo necesiten.

En este momento, todos y cada uno de los ejes del grafo disponen de tres parámetros característicos, 'weight'=1.0, 'probability'=0.0 y el valor 'pheromone' que ha sido calculado en el paso anterior. Es necesario nombrar que las soluciones otorgadas por las iteraciones de inicialización son almacenadas a fin de que éstas pudieran ser usadas posteriormente en caso de que se necesitaran. El siguiente paso en el desarrollo del programa es realizar una serie de iteraciones con el fin de que éstas mejoren al conjunto de iteraciones de inicialización.

Para ello, el proceso de selección del recorrido en dicho conjunto de iteraciones finales sigue el siguiente mecanismo. En cada iteración, el programa se ha de ir repitiendo hasta que todos los ejes hayan sido visitados, momento en el cual, al igual que ocurría en los programas anteriores, se aplicará el algoritmo 'shortest path' a fin de volver al punto de partida. Cada vez que un eje es atravesado, el valor 'weight' en dicho eje será multiplicado por 10.0. Hasta éste momento, el procedimiento no difiere en absoluto de los programas anteriores pero, a partir de aquí, el proceso de selección de un eje u otro difiere considerablemente con el

procedimiento seguido por las iteraciones de inicialización.

Siempre se procede a seleccionar aquel eje que no haya sido recorrido todavía, es decir, aquel eje que tenga como parámetro 'weight' un valor 1.0. En caso de que exista más de un eje incidente al nodo en el que la iteración se encuentra que todavía no ha sido recorrido, la decisión de seleccionar uno u otro se va a realizar en base a una función probabilística dependiente del valor del parámetro 'pheromone' de cada uno de ellos tal y como se muestra en la Ecuación 6 donde i es el eje a tener en consideración y α es un factor que incrementa o reduce la importancia de la información obtenida en las anteriores iteraciones a la hora de otorgar la probabilidad de que un eje sea seleccionado. Por el momento, dicho parámetro toma valor $\alpha = 1$.

$$new\_weight_i = (pheromone_{max} - pheromone_{min}) + (pheromone_i - pheromone_{min})$$
$$probability_i = \frac{(new\_weight_i)^\alpha}{\sum_{i=1}^{n}(new\_weight_i)^\alpha}$$

Ecuación 6: Cálculo de probabilidad en el proceso de selección

A modo de ejemplo se añade el siguiente caso en el que el número de ejes adyacentes n a que todavía no han sido recorridos es tres y el valor del parámetro 'pheromone' de cada uno de ellos es respectivamente 210, 215 y 230.

$$n = 3$$
$$pheromone\ values = 210, 215, 230$$
$$new_{weight_1} = [(230 - 210) + (210 - 210)] = 20$$
$$new_{weight_2} = [(230 - 210) + (215 - 210)] = 25$$
$$new_{weight_3} = [(230 - 210) + (230 - 210)] = 40$$
$$probability_1 = \frac{20}{20 + 25 + 40} = 0.235$$
$$probability_2 = \frac{25}{20 + 25 + 40} = 0.294$$
$$probability_3 = \frac{40}{20 + 25 + 40} = 0.470$$

Ecuación 7: Ejemplo en el caso de tres ejes adyacentes y $\alpha = 1$

Una vez obtenida dicha probabilidad y almacenada en el parámetro 'probability' de cada uno de ellos se procede a la selección de cada uno de ellos mediante la generación de un número aleatorio siguiendo el mismo procedimiento descrito en la Ecuación 4. Con dicho procedimiento de selección, se consigue otorgar más probabilidad de selección a aquellos ejes que tienen un valor de 'pheromone' más alto.

En caso de que todos los ejes vecinos hayan sido ya recorridos, la selección de uno u otro como el siguiente eje a ser recorrido se realizará también en función del parámetro 'pheromone' con el mismo mecanismo que el indicado en la Ecuación 5 para calcular la probabilidad y su posterior procedimiento de selección indicado en la Ecuación 6 con la única diferencia que, en éste caso, los ejes a considerar serán todos los adyacentes al nodo en el cual la iteración se encuentre.

CONCLUSIONES:

El programa ha sido aplicado al grafo de la Universidad de Rhode Island con un número constante de 1000 iteraciones finales, es decir, las cuales utilizan los valores de 'pheromone' calculados, y se ha variado tanto el número de iteraciones de inicialización como la fórmula de actualización del parámetro de 'pheromone'. Ésta última modificación ha cambiado la fórmula dada en la Ecuación 5 por la misma pero con un valor

multiplicativo de 10.0. El propósito de ésta modificación es que exista todavía más diferencia entre la calidad de las diferentes iteraciones de inicialización con la mejor de todas ellas. Los resultados se muestran en la Tabla 9.

Programa 2_8_2011				
Valor multiplicativo	10			
Nº de iteraciones de inicialización	100	1000	100	1000
Actualización de feromonas	$e^{-4*(1-\left(\frac{longitud_X}{\min(longitud_1^m)}\right)^2)}$		$e^{-10*(1-\left(\frac{longitud_X}{\min(longitud_1^m)}\right)^2)}$	
Promedio	3862.6	5811.3	4864.1	4905
Derivación estándar	1856.3	3613.8	2607.9	2646.1
Mínimo	1093	1103	1309	1327
Máximo	18595	28974	18812	20245

Tabla 9: Resultados para el programa 2_8_2011.

Como se puede observar, el hecho de variar el número de iteraciones de inicialización no modifica en esencia los resultados finales, sobre todo en el caso de usar la fórmula más restrictiva de actualización del parámetro 'pheromone' a pesar de que, obviamente, dicho valor de actualización de feromonas que cada eje recibe es mucho mayor en el caso de que haya 1000 iteraciones de inicialización en lugar de 100. Además, también se puede ver como el uso de una u otra fórmula en el momento de actualizar el parámetro 'pheromone' tampoco lleva a una conclusión determinada ya que, en el caso de 100 iteraciones de inicialización el valor tanto de promedio como de desviación estándar mejoran al aplicar la fórmula más restrictiva pero, en el caso de 1000 iteraciones de inicialización ambos valores son mejores al usar la fórmula menos restrictiva.

En general, el hecho de que los valores obtenidos de desviación estándar sean tan elevados en comparación con los obtenidos en el programa anterior hace que se considere totalmente inapropiado extraer conclusiones definitivas de éste conjunto de pruebas realizadas ya que, la enorme variación de los resultados obtenidos guía a pensar que el problema en dicho programa es debido a un inapropiado procedimiento de selección de los ejes más que al propio concepto de aplicación del parámetro 'pheromone'. Por ello, no se puede llegar a la conclusión de que dicho procedimiento de trabajo utilizando el parámetro 'pheromone' actúe de una forma apropiada o inapropiada en cuanto a la calidad de las soluciones finales se refiere.

A DÍA 2_10_2011:

A partir de las conclusiones extraídas en el programa anterior, se considera necesario y apropiado el modificar el procedimiento de selección de los ejes en aquellas iteraciones finales que cuentan con la información correspondiente a la calidad de las iteraciones de inicialización.

En el presente programa se mantiene tanto el procedimiento de trabajo de las iteraciones de inicialización, ya que éste se consideró apropiado en las correspondientes pruebas realizadas con el mismo, como el valor de cálculo del parámetro 'pheromone' introducido en el programa anterior. Por tanto, la modificación del programa va a ser realizada tan solo en el conjunto de las iteraciones que ya cuentan con la información de las iteraciones de inicialización.

Al igual que ocurría en el programa anterior, el grafo va a ser recorrido hasta que todos los ejes hayan sido visitados, momento en el cual se aplicará el algoritmo 'shortest path' con el fin de volver al eje inicial. Por otra parte, el valor del parámetro 'weight' de cada eje seguirá siendo multiplicado por 10.0 cada vez que éstos sean atravesados en una dada iteración. Además, como ya se comentó anteriormente, el cálculo del valor del parámetro 'pheromone' se realizará de la misma forma indicada anteriormente en la Ecuación 5.

El único cambio realizado en comparación con el programa anterior será el procedimiento de selección de un eje en el caso de que todos y cada uno de ellos ya hayan sido visitados. En lugar de acudir al parámetro 'pheromone' como se hacía en el programa anterior, usando la Ecuación 6 para calcular la probabilidad de acudir a uno u a otro, en éste momento se realizará dicha selección de la misma manera que se realizaba en las iteraciones iniciales, es decir, utilizando el valor del parámetro 'weight' para calcular la probabilidad de selección de dichos ejes, tal y como se introdujo en la Ecuación 3 y en la Ecuación 4.

CONCLUSIONES:

Con el fin de poder realizar una comparación objetiva de los resultados obtenidos con ésta variación del programa, se ha aplicado el mismo de la misma forma que se realizó el anterior, con un conjunto 1000 iteraciones finales en el grafo de la Universidad en Rhode Island. Dichos resultados se muestran en la Tabla 10.

Programa 2_10_2011				
Valor multiplicativo	10			
Nº de iteraciones de inicialización	100	1000	100	1000
Actualización de feromonas	$e^{-4*(1-\left(\frac{longitud_X}{\min(longitud_1^m)}\right)^2)}$		$e^{-10*(1-\left(\frac{longitud_X}{\min(longitud_1^m)}\right)^2)}$	
Promedio	1484.6	1509.4	1480.4	1492.5
Derivación estándar	393.4	388.5	407.1	379.9
Mínimo	679	743	677	736
Máximo	3381	3379	3911	3090

Tabla 10: Resultados para el programa 2_10_2011

Tras el análisis de dichos resultados se puede ver claramente como los resultados obtenidos son realmente satisfactorios. Los valores promedios se han reducido cerca de un 40% en comparación con los obtenidos en el programa 1_25_2011, el cual no utilizaba el factor 'pheromone', los cuales son mostrados en la Tabla 8 para valores del valor multiplicativo de 10.0. Además, los valores de desviación estándar han mejorado en cerca del 60% lo que realmente indica que el procedimiento de trabajo desarrollado en ésta modificación del programa es realmente adecuado.

Como último punto a estudiar, se puede observar cómo tanto las modificaciones del número de iteraciones de inicialización como el uso de una u otra fórmula de actualización del parámetro 'pheromone' no llevan a variaciones significativas de las soluciones obtenidas.

A DÍA 3_10_2011:

A pesar de que los resultados obtenidos a partir del programa anterior son realmente buenos y alentadores, pudiéndose incluso considerar como definitivos, se va a modificar dicho programa con el fin de intentar mejorar todavía más dichos resultados.

Puesto que el aspecto fundamental que hace del programa anterior una mejora tan sustancial de los resultados es el hecho de usar la información obtenida en previas iteraciones en la selección de los nuevos recorridos a través del parámetro 'pheromone', se considera como una posible mejora del mismo el hecho de utilizar todavía más información referente a soluciones anteriores a la hora de computar las nuevas.

Al igual que sucede con los algoritmos de optimización basados en colonias de hormigas, en los que se simula el proceso de evaporación de feromonas y el depósito de las mismas con el propósito de mostrar la conveniencia de uno u otro recorrido, en éste programa se actualizan los valores del parámetro 'pheromone' de una manera continua y constante con el objetivo de ir mejorando la calidad de la información previa disponible. Gracias a ello se simulan aproximadamente ambos mecanismos al mismo tiempo, es decir, tanto el proceso de evaporación como el de depósito de feromonas ya que la información es actualizada poco a poco por lo que se marca la conveniencia de aquellos nuevos recorridos reduciendo además la de los antiguos y, puesto que teóricamente los nuevos mejoran a los antiguos, dicha información se va haciendo más valiosa a cada una de las actualizaciones.

Para ello, en primer lugar, se realizan una serie de iteraciones de inicialización y el correspondiente cálculo del valor del parámetro 'pheromone' tal y como se mostró en los anteriores programas con el propósito de obtener una información inicial suficiente y valiosa.

Una vez éste proceso de inicialización ha sido finalizado, la idea es realizar un conjunto de iteraciones que tengan como base dicha información y, al finalizar, volver a realizar la actualización del parámetro 'pheromone' usando para ello todos los recorridos que hayan surgido hasta el momento, es decir, tanto aquellos de inicialización como los del primer bloque de iteraciones. Una vez ya se hayan actualizado dichos valores de 'pheromone' se realiza otro conjunto de nuevas iteraciones pero en éste caso tomando como información los recientes valores del parámetro 'pheromone', y al finalizar el nuevo conjunto, se volverán a actualizar dichos valores de nuevo teniendo en cuenta todos los recorridos disponibles. Éste proceso se repetirá un determinado número de veces.

Por otra parte, es necesario nombrar que, a pesar de que el proceso óptimo sería que dicha actualización de feromonas se realizara lo más frecuentemente posible, es decir, al final de cada iteración comparar dicho valor con los anteriores y actualizar el parámetro, se considera que éste método retrasaría enormemente el tiempo de ejecución del programa por lo que, debido a ello, dichas actualizaciones se realizan al final de cada bloque de iteraciones en lugar de al finalizar cada una de ellas.

CONCLUSIONES:

Tras aplicar dicho programa al ya conocido grafo de la Universidad de Rhode Island se han obtenido los datos mostrados en la Tabla 11. Como se puede observar, se han realizado 15 bloques diferenciados más el de inicialización estando cada uno de ellos compuesto por 50 iteraciones.

Programa 3_10_2011								
Bloque	INIZ.	1	2	3	4	5	6	7
Promedio	2377.7	1462.5	1495.5	1400.7	1415.7	1426.5	1452.2	1357.5
Derivación estándar	531.9	326.7	495.2	420.9	452.2	394.9	394.9	352.58
Mínimo	1526	970	744	792	708	750	869	845
Máximo	3842	2666	3554	2578	3605	2399	2603	2394
Bloque	8	9	10	11	12	13	14	15
Promedio	1445.4	1564.9	1390.6	1429.7	1382.4	1330.4	1429.7	1362.2
Derivación estándar	385.4	491.9	423.9	490.2	349.6	411.7	499.5	331.8
Mínimo	834	782	838	696	864	813	744	843
Máximo	2697	3118	2994	3369	2244	3182	3070	2413

Tabla 11: Resultados para el programa 3_10_2011

Como se puede analizar de los resultados obtenidos, una vez tanto el promedio como la derivación estándar mejoran enormemente en el paso del bloque de inicialización, en el que no se usan feromonas, al primer bloque, el cual ya usa dicho parámetro en su decisión, los resultados de los siguientes bloques no proporcionan una mejora significativa ya que tanto valores mínimos, promedios y derivaciones estándar van variando de forma aleatoria entre ellos aunque siempre mostrando unos resultados realmente buenos, al igual que sucedía con el programa anterior. Se puede observar como, por ejemplo, la diferencia en los resultados al usar como información previa 500 recorridos previos, como es el caso de la 10ª iteración, no es mejor que el hacerlo teniendo en cuenta tan solo 150 recorridos, como es el caso de la 3ª iteración.

Por este motivo, se llega a la conclusión de que ya se ha alcanzado el límite de mejora usando este procedimiento de decisión basado en el parámetro 'pheromone'.

A DÍA 3_17_2011

Tal y como se comentó en la sección '3. Puntos clave en el mantenimiento invernal de la red viaria', en la gran mayoría de las operaciones de mantenimiento invernal que implican el ruteo de vehículos, tales como el esparcimiento de químicos y abrasivos, la recogida de nieve y, en especial, la retirada de la misma con máquinas quitanieves, es necesario establecer un determinado nivel de servicio a cada una de las carreteras que conforman el conjunto viario a tratar dependiente de la importancia que cada una de éstas tiene. La motivación de realizar ésta clasificación se debe, como ya se nombró anteriormente, a las necesidad de que un subconjunto de carreteras deban de ser limpiadas u operadas en un periodo determinado de tiempo debido a la mayor importancia de éstas dentro del conjunto total de la red viaria. Dicha importancia puede ser basada en diversos factores como motivos políticos, económicos o simplemente, necesidad de un servicio más rápido.

Hasta el momento, en todos los programas anteriores se ha considerado que se debían visitar todos y cada uno de los ejes al menos una vez, es decir, se ha estado solucionando el conocido 'Chinese Postman Problem', sin embargo, a partir de éste momento, los algoritmos desarrollados resolverán una variante de dicho problema llamada 'Rural Chinese Postman Problem' en el cual tan solo un conjunto de ejes deben de ser recorridos obligatoriamente.

Con el fin de implementar dicho problema, el primer cambio obvio a realizar es añadir a todos los ejes un nuevo parámetro el cual debe indicar si dicho eje se encuentra en el subconjunto de ejes a visitar, es decir, si dicho eje ha de ser recorrido obligatoriamente. A dicho parámetro se la otorgado el nombre de 'mandatory' y puede tener bien valores de 0, en el caso de que el eje no se encuentre en el subconjunto de ejes a visitar o de

1 en caso contrario. Una vez éste parámetro ha sido añadido, se ha cambiado la condición principal por la cual una hormiga debía de finalizar su proceso de recorrido en el grafo. Anteriormente, un recorrido se consideraba terminado cuando todos los ejes habían sido recorridos al menos una vez, sin embargo, de aquí en adelante, la condición para finalizar dicho recorrido es que todos los ejes con el parámetro 'mandatory=1' hayan sido recorridos al menos una vez.

Por otra parte, en éste programa y en los siguientes, en los cuales se trata de resolver el 'Rural Chinese Postman Problem', la información adquirida en las iteraciones previas y almacenada en el parámetro 'pheromone' es mucho más importante que en el caso del 'Chinese Postman Problem' ya que, al no ser necesario recorrer todos los ejes del grafo, la información obtenida en previas iteraciones acerca del recorrido realizado adquiere mucho más valor. Por esa razón principalmente, se ha decidido modificar ligeramente el procedimiento de selección de probabilidad de los ejes en función de el parámetro 'pheromone' que se daba en la Ecuación 6. A partir de ahora, el parámetro α que actúa como potenciador del valor de la información previa depositada en el parámetro 'pheromone' pasa a ser 2 en lugar de 1 como hasta ahora. A fin de mostrar como dicho cambio afecta en las probabilidades de que un eje u otro sea seleccionado, se realizan los cálculos para los mismos valores de 'pheromone' que se dieron en la Ecuación 7: Ejemplo en el caso de tres ejes adyacentesEcuación 7, viendo cómo se incrementan las diferencias de probabilidad de selección de un eje frente a otro con respecto a las probabilidades obtenidas usando $\alpha = 1$.

$$n = 3$$
$$pheromone\ values = 210, 215, 230$$
$$new_{weight_1} = [(230 - 210) + (210 - 210)] = 20$$
$$new_{weight_2} = [(230 - 210) + (215 - 210)] = 25$$
$$new_{weight_3} = [(230 - 210) + (230 - 210)] = 40$$
$$probability_1 = \frac{(20)^2}{(20)^2 + (25)^2 + (40)^2} = 0.152$$
$$probability_2 = \frac{(25)^2}{(20)^2 + (25)^2 + (40)^2} = 0.238$$
$$probability_3 = \frac{(40)^2}{(20)^2 + (25)^2 + (40)^2} = 0.610$$

Ecuación 8: Ejemplo en el caso de tres ejes adyacentes y $\alpha = 2$

A pesar de los cambios introducidos, el procedimiento general de trabajo del programa no varía de forma sustancial con respecto a los programas anteriores. En primer lugar, se realiza un bloque de iteraciones de inicialización con el mismo procedimiento que el seguido hasta ahora, es decir, simplemente seleccionando los ejes de forma probabilística inversamente proporcional al parámetro 'weight', el cual se va incrementando cada vez que un eje es recorrido, con la única diferencia de que la condición de finalización de la iteración ha cambiado, siendo ahora necesario que tan solo los ejes con 'mandatory'=1 hayan sido visitados. A partir de ahí, se calcula el parámetro 'pheromone' de la misma forma que la indicada en la Ecuación 5 con la información obtenida de las iteraciones de inicialización. Con dichos valores ya calculados, se realizan diferentes bloques de iteraciones usando el mismo procedimiento que se ha llevado a cabo hasta ahora, es decir, realizando la selección de entre aquellos ejes que todavía no han sido recorridos mediante la información almacenada en el parámetro 'pheromone' aunque, como ya se nombró anteriormente, usando $\alpha = 2$ en el cálculo de dicha probabilidad de selección mostrada en la Ecuación 6. Además, tal y como se viene desarrollando hasta el momento, en el caso de que todos los ejes adyacentes a un nodo ya hayan sido visitados, el procedimiento de selección probabilística se basará en el parámetro 'weight' que nos indica cuantas veces un eje ya ha sido visitado, eliminando de ésta forma gran parte de los bucles que se pudieran

formar.

En cuanto a los valores de actualización del parámetro 'pheromone' se considera como una medida más apropiada en éste caso el considerar tan solo la información obtenida en el último bloque de iteraciones realizadas. La enorme importancia que en el 'Rural Chinese Postman Problem' tiene el valor del parámetro 'pheromone' a la hora de realizar dicha selección de los nuevos recorridos, ya que nos indicará la conveniencia de seleccionar uno u otro, hace necesario que dicha información sea de la mayor calidad posible. Por ello, puesto que teóricamente, los resultados mejoran de un bloque de iteraciones a otro, se va a utilizar tan solo el último de los bloques realizados con el propósito de usar tan solo la mejor información disponible. Éste mecanismo de actualización tiene el mismo propósito que el proceso de evaporación de feromonas usado en los algoritmos de optimización basados en colonias de hormigas estudiados en la sección '6.2. Algoritmos de optimización basados en colonias de hormigas (ACO)'.

CONCLUSIONES:

Con el propósito de analizar los resultados proporcionados con el programa en un entorno lo más parecido a la realidad, se ha aplicado dicho programa al grafo de la universidad de Rhode Island. Puesto que el programa debe de contemplar qué ejes del grafo es obligatorio atravesar y cuáles no, se ha estimado como un valor apropiado el que alrededor de un 25% de los ejes del campus sean obligatorios, es decir, tengan el parámetro 'mandatory'=1. La selección de qué ejes de los 257 ejes del grafo deben ser obligatorios se ha realizado de manera totalmente aleatoria ya que es considerado simplemente un ejemplo a fin de poder aplicar los distintos programas.

A pesar de ello, es obvio notar que, en posteriores aplicaciones reales del programa, sería obligatorio realizar un estudio detallado de cuales son aquellos ejes que necesitan de un mayor nivel de servicio. Para ello, como ya se nombró en el apartado '3.2.1. Nivel de servicio', diversos factores deberían ser considerados y analizados en profundidad. Por tanto, a modo de prueba se han seleccionado 74 ejes como obligatorios de una manera aleatoria, los cuales, van a permanecer siendo obligatorios en las próximas variaciones del programa a fin de que sea posible realizar una comparación adecuada de los resultados obtenidos entre cada uno de los diferentes programas.

Tal y como se nombró en la sección '2.5. Herramientas y técnicas utilizadas', la gran cantidad de datos a calcular y la carga de memoria que éstos requieren, hacen que sea totalmente necesaria la utilización del ordenador de dos núcleos del laboratorio. Por ello, todos los resultados mostrados a continuación han sido obtenidos mediante dicho ordenador con el propósito de agilizar la obtención de los mismos.

En un primer lugar, tal y como se muestra en la Tabla 12 , se ha ejecutado el programa con un bloque de inicialización de 500 iteraciones, realizando posteriormente 11 diferentes bloques de 1000 iteraciones cada uno.

Programa 3_17_2011						
Bloque	INIZ.	1	2	3	4	5
Iteraciones	500	1000	1000	1000	1000	1000
Promedio	2035.9	1269.4	1164.4	1125.9	1155.0	1125.3
Derivación estándar	614.4	380.5	360.7	374.4	379.1	360.5
Mínimo	1062	568	532	453	496	469
Máximo	4508	3787	3040	4520	3035	3172
Bloque	6	7	8	9	10	11
Iteraciones	1000	1000	1000	1000	1000	1000
Promedio	1116.2	1113.9	1123.5	1135.0	1111.9	1119.7
Derivación estándar	339.6	352.9	355.9	354.0	333.3	360.1
Mínimo	515	490	489	523	487	506
Máximo	2807	3298	2688	2905	2736	3132

Tabla 12: Resultados para el programa 3_17_2011 con bloques de 1000 iteraciones

Como se puede observar, los valores de derivación estándar permanecen en torno a 360, lo cual se puede considerar como buenos resultados y las diferencias entre el valor promedio de cada bloque de iteraciones son totalmente despreciables, estando entorno a un valor de 1100, lo cual también indica que el programa está proporcionando resultados más que aceptables. Por otra parte, tal y como se observa en el Gráfico 9 , se puede ver como los valores mínimos de cada bloque de iteraciones no varían tan apenas. De la misma forma, se observa como éstos valores proporcionados son realmente buenos, llegando incluso a realizar en el bloque 3 un recorrido de 453. Como conclusión de éstos resultados se puede llegar a decir que el programa desarrollado ya ha alcanzado un nivel muy bueno de soluciones, teniendo que modificarse el mismo si se desea intentar conseguir todavía mejores resultados.

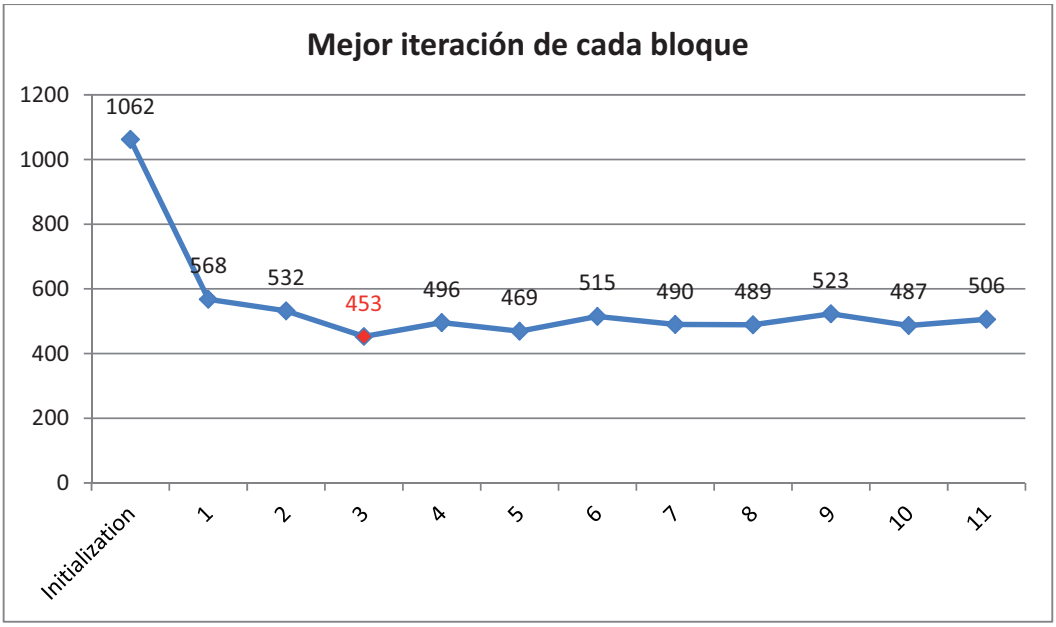

Gráfico 9: Valores mínimos de cada bloque de 1000 iteraciones para el programa 3_17_2011

A fin de asegurarse de que las conclusiones obtenidas gracias a las pruebas realizadas anteriormente son correctas, y, por tanto, el programa ya ha alcanzado su propio límite de mejora, ya que no se aprecia tan apenas modificación en los valores mínimos de las iteraciones de cada bloque, es necesario volver a aplicar el programa de una manera distinta para de ésta forma comprobar si éstos valores mínimos realmente se mantienen aproximadamente constantes a lo largo de los diferentes bloques de iteraciones.

Además de comprobar que el programa ya ha alcanzado un límite de mejora, se va a analizar qué procedimiento es más conveniente, realizar grandes bloques de iteraciones y posteriormente actualizar el valor del parámetro 'pheromone' o bien, realizar más bloque s de iteraciones pero con un número más reducido de iteraciones, por lo que dicho parámetro 'pheromone' se actualizará más constantemente pero con menor cantidad de información.

Para ello, se ha aplicado el programa de nuevo haciendo un bloque de inicialización de tan solo 100 iteraciones y, posteriormente, 46 nuevos bloques de otras 100 iteraciones cada uno. De ésta forma, los valores de 'pheromone' se van actualizando de una manera más constante teniendo solo en cuenta las 100 últimas iteraciones realizadas, a diferencia de las 1000 iteraciones que se tenían en cuenta en la prueba realizada anteriormente. Los resultados, así como sus valores medios se pueden observar en la Tabla 13.

Programa 3_17_2011								
Bloque	INIZ.	1	2	3	4	5	6	7
Promedio	1957.8	1265.0	1193.3	1146.5	1205.1	1152.5	1185.1	1240.8
Derivación estándar	585.4	387.0	404.5	327.9	325.0	311.5	365.7	420.0
Mínimo	1136	616	651	616	570	571	648	669
Máximo	3970	2563	2716	2129	2281	2309	2434	3398
Bloque	8	9	10	11	12	13	14	15
Promedio	1254.4	1157.5	1200.1	1165.9	1210.7	1160.9	1220.1	1244.1
Derivación estándar	396.9	411.8	439.4	348.2	388.9	340.9	355.9	357.3
Mínimo	649	617	591	611	579	689	700	536
Máximo	2983	2882	3083	2334	2527	2241	2173	2327
Bloque	16	17	18	19	20	21	22	23
Promedio	1151.9	1168.2	1272.2	1200.9	1189.7	1107.4	1173.8	1124.8
Derivación estándar	375.8	333.0	420.8	349.8	402.4	351.2	335.6	352.8
Mínimo	640	673	688	711	526	537	558	628
Máximo	2352	2433	2780	2606	2543	2939	2106	2454
Bloque	24	25	26	27	28	29	30	31
Promedio	1096.9	1186.6	1102.3	1094.6	1060.5	1212.6	1157.1	1153.9
Derivación estándar	330.1	334.5	287.3	295.1	317.4	332.0	338.3	319.0
Mínimo	550	502	542	625	537	602	639	631
Máximo	2312	2572	1998	1849	2495	2160	2412	2600

Bloque	32	33	34	35	36	37	38	39
Promedio	1231.9	1192.5	1119.4	1155.8	1252.5	1181.7	1109.5	1135.8
Derivación estándar	375.3	334.1	355.2	332.2	441.5	405.1	345.6	364.0
Mínimo	641	661	518	675	569	589	431	564
Máximo	2399	2416	2766	2423	3471	2910	2046	2471

Bloque	40	41	42	43	44	45	46	***PROM.***
Promedio	1079.9	1120.1	1147.2	1162.1	1137.2	1247.0	1109.1	***1170.4***
Derivación estándar	336.3	307.1	342.0	294.1	269.5	398.9	319.0	***353.8***
Mínimo	600	521	683	717	572	686	583	***606.8***
Máximo	2749	2071	2399	2607	1853	2547	2569	***2493.2***

Tabla 13: Resultados para el programa 3_17_2011 con bloques de 100 iteraciones

Una vez analizados éstos datos se puede observar cómo, en primer lugar, los resultados obtenidos en cuanto a los valores mínimos obtenidos son peores en éste caso en el caso estudiado anteriormente, el cual se mostraba en la Tabla 12, con grandes bloques de iteraciones y actualizaciones con mayor cantidad de datos que en la presente prueba. Por ello, se puede considerar como adecuado que en los próximos programas desarrollados dichas actualizaciones se lleven a cabo con gran cantidad de información y con grandes bloques de iteraciones.

Por otra parte, tal y como se puede observar en el Gráfico 10, se comprueba que, efectivamente, el programa desarrollado ha alcanzado su límite de mejora ya que los valores mínimos de cada bloque de iteraciones tan apenas varían de un bloque a otro.

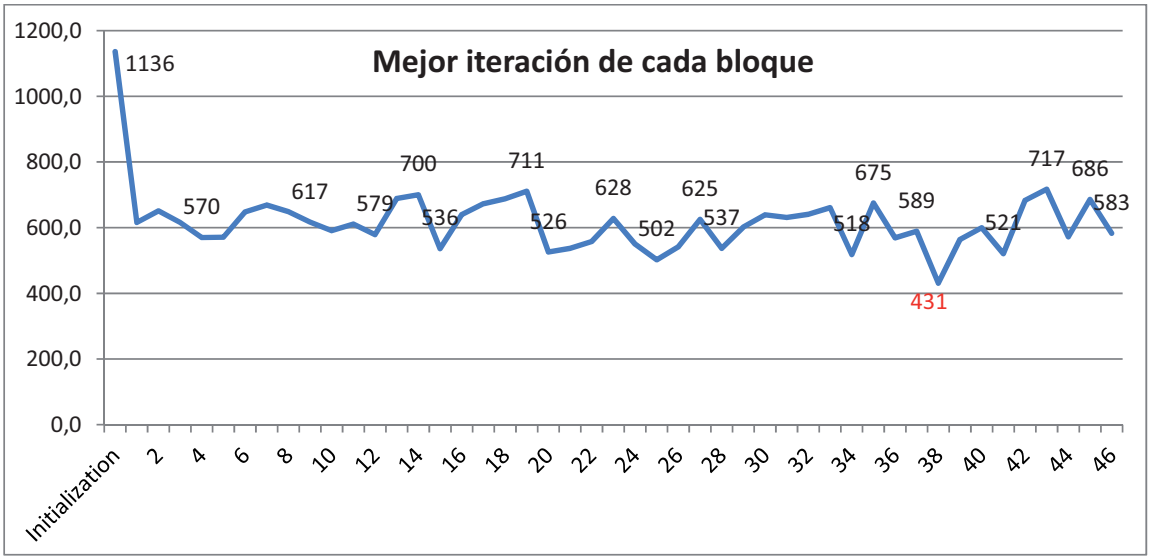

Gráfico 10: Valores mínimos de cada bloque de 100 iteraciones para el programa 3_17_2011

A DÍA 4_4_2011

Puesto que el programa realizado anteriormente nos ofrece unos resultados realmente buenos, éste constituirá la base a partir de la cual se realizará la presente modificación.

Tal y como se estudió anteriormente, el programa ya ha llegado a un límite máximo de mejora por lo que, las modificaciones a realizar en el mismo deben simplemente añadir nuevas restricciones al programa ya desarrollado y, en todo caso, añadir mejoras locales a las soluciones ya obtenidas.

Por ello, como restricción a añadir al programa, se introduce un límite máximo de longitud que cada iteración puede tener. En el caso de que una iteración sobrepase ese límite máximo, ésta no será tenida en cuenta en el proceso de actualización de feromonas por lo que, de ésta forma, se asegura que la información a partir de la cual las feromonas son actualizadas es una información adecuada y valiosa, no habiendo, como anteriormente, diferentes iteraciones que añadían recorridos realmente negativos para la mejora de las soluciones.

Para ello, se ha considerado adecuado el establecer dicho valor máximo de cada iteración como un valor de $1{,}5 * Best\_so\_far$, donde $Best_so_far$es el recorrido más corto hasta el momento.

A fin de no ralentizar el programa enormemente, en lugar de analizar al final de cada iteración si ésta es mejor que la mejor encontrada hasta el momento ($Best\_so\_far$), al acabar cada bloque de iteraciones se comprueba si alguna de las iteraciones de dicho bloque es mejor que la que hasta el momento se consideraba la mejor y, en caso de que así sea, se aplica la nueva limitación de longitud al siguiente bloque de iteraciones.

En cuanto al proceso llevado a cabo en cada bloque de iteraciones, salvo en el primer bloque de iteraciones de inicialización, justo antes de iniciar el mismo se establece cual es el límite máximo que una iteración puede alcanzar, siendo dicha limitación basada, como ya se comentó anteriormente, en la mejor iteración hasta el momento. Una vez establecido dicho límite, se comienzan a hacer iteraciones de forma que cuando alguna de ellas supera dicho límite, ésta es anulada y se vuelve a comenzar la misma hasta que se consigue que la longitud de la misma se encuentre por debajo del límite establecido, momento en el cual, dicho recorrido es almacenado en la memoria. Éste procedimiento se realiza tantas veces como sea necesario hasta que se consigue un bloque de X iteraciones en las que todas ellas se encuentran por debajo del límite establecido. En ése momento, se procede a la actualización de feromonas para el siguiente bloque con el conjunto de iteraciones almacenadas del último bloque realizado. Una vez esto finaliza, se repite de nuevo el proceso, analizando si alguna de las soluciones del último bloque es mejor que la mejor solución que había hasta ahora y, en caso de que la haya, modificando el límite máximo para el siguiente bloque de iteraciones.

CONCLUSIONES:

Tal y como se realizó en el programa anterior, ase va a realizar la prueba usando dos metodologías distintas, aplicando el programa con pocos bloques pero de un gran número de iteraciones y, por otra parte, aplicando el mismo con muchos bloques de iteraciones pero de pocas iteraciones cada uno. De ésta forma se puede comprobar exactamente la diferencia en los resultados entre el programa anterior y el presente además de asegurar las conclusiones que se obtuvieron del estudio de los resultados del programa anterior.

En la Tabla 14 , se muestra el caso de un bloque de inicialización de 500 iteraciones y 11 posteriores bloques de 1000 iteraciones cada uno. Además, se puede analizar al igual que en el programa anterior, como los valores mínimos de cada bloque de iteraciones van variando en el Gráfico 11.

Programa 4_4_2011						
Bloque	INIZ.	1	2	3	4	5
Iteraciones	500	1000	1000	1000	1000	1000
Promedio	2050.1	1051.1	765.3	617.4	614.9	609.6
Derivación estándar	610.3	178.2	82.5	42.7	45.7	42.3
Mínimo	898	587	448	457	444	413
Máximo	4889	1346	880	672	672	666
Bloque	6	7	8	9	10	11
Iteraciones	1000	1000	1000	1000	1000	1000
Promedio	575.92	573.84	564.30	563.14	563.98	545.53
Derivación estándar	34.72	37.35	34.64	34.42	34.26	31.24
Mínimo	432	404	402	405	388	406
Máximo	619	619	606	603	603	582

Tabla 14: Resultados para el programa 4_4_2011 con bloques de 1000 iteraciones

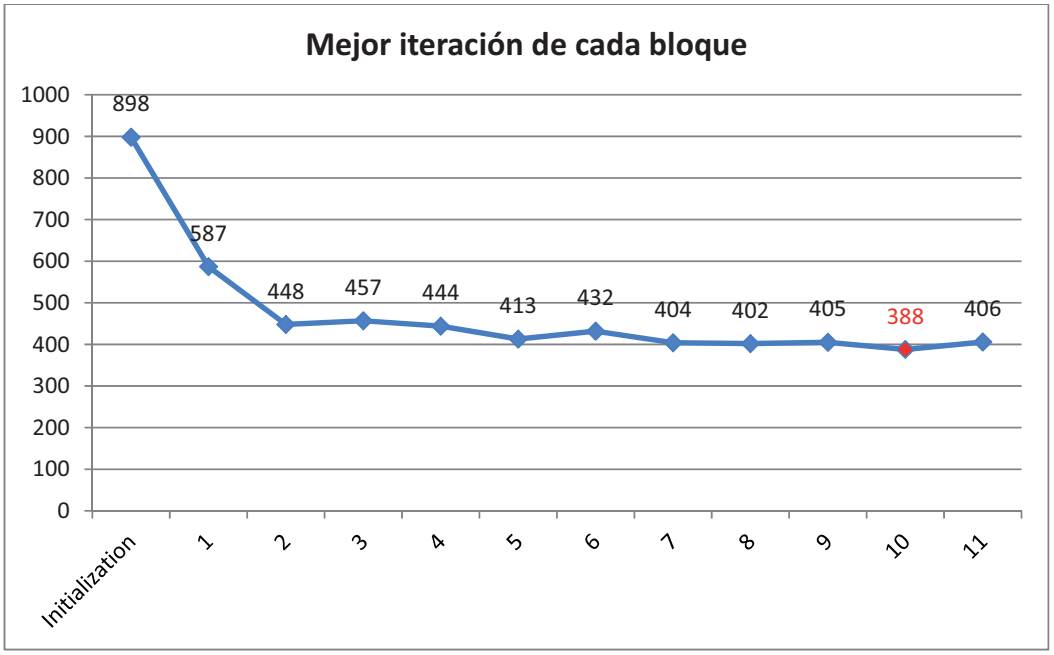

Gráfico 11: Valores mínimos de cada bloque de 1000 iteraciones para el programa 4_4_2011

En comparación con los resultados obtenidos en el programa anterior al aplicar el mismo número de bloques de iteraciones y el mismo número de iteraciones en cada uno de ellos, Tabla 12, se puede observar como los resultados obtenidos son realmente buenos, consiguiendo pasar de un promedio en cada bloque de alrededor a 1100 en el programa anterior a unos valores alrededor de 600 en el presente programa. Además, en cuanto a la desviación estándar se refiere, los resultados obtenido por este programa muestran un valor decreciente, desde alrededor de 80 hasta incluso valores de 30 en los últimos bloques de iteraciones, lo cual, en

comparación con el promedio obtenido en el programa realizado el día 3_17_2011, alrededor de 360, muestra una mejora increíble. Por otra parte, se observa como el valor mínimo desciende globalmente en todos los bloques de iteraciones siendo en este caso el mejor recorrido 388, un valor realmente bueno y adecuado teniendo en cuenta el número de ejes que se han considerado como obligatorios de recorrer.

Por otra parte, en la Tabla 15, se muestran los datos obtenidos al aplicar el programa con 46 bloques de iteraciones de 100 iteraciones cada uno además de un bloque de inicialización de también 100 iteraciones.

Programa 4_4_2011								
Bloque	INIZ.	1	2	3	4	5	6	7
Promedio	1893.7	1106.0	875.0	777.1	708.1	701.5	691.8	701.3
Derivación estándar	461.8	202.9	112.9	88.7	50.1	54.6	56.4	53.1
Mínimo	974	715	597	518	549	565	535	560
Máximo	3116	1449	1072	895	777	777	773	775
Bloque	8	9	10	11	12	13	14	15
Promedio	699.0	676.9	680.4	660.1	623.4	620.2	593.0	585.7
Derivación estándar	57.8	55.4	57.0	55.2	39.7	49.5	35.2	40.2
Mínimo	497	497	486	450	481	423	462	433
Máximo	773	745	745	728	675	675	633	634
Bloque	16	17	18	19	20	21	22	23
Promedio	590.6	586.8	594.4	583.2	588.7	590.4	592.8	587.7
Derivación estándar	36.7	38.0	35.2	36.3	32.2	38.1	32.3	44.5
Mínimo	431	488	469	485	486	453	488	439
Máximo	634	634	634	634	634	634	634	634
Bloque	24	25	26	27	28	29	30	31
Promedio	589.3	584.4	587.5	584.1	587.9	579.6	583.0	554.0
Derivación estándar	37.2	44.6	37.8	37.6	40.5	44.0	47.2	32.7
Mínimo	438	438	476	450	449	444	395	431
Máximo	634	634	634	632	634	634	634	591
Bloque	32	33	34	35	36	37	38	39
Promedio	550.5	555.5	555.7	552.3	555.7	557.6	552.6	558.5
Derivación estándar	31.9	30.0	32.4	32.1	33.1	29.7	38.4	29.0
Mínimo	467	446	453	441	433	439	419	463
Máximo	592	592	592	592	592	592	592	592
Bloque	40	41	42	43	44	45	46	***PROM.***
Promedio	549.2	552.6	552.0	552.0	554.7	558.6	556.0	616.9
Derivación estándar	30.0	30.5	32.0	34.8	33.0	27.2	34.2	45.7
Mínimo	462	461	449	427	453	482	464	473.6
Máximo	592	592	589	592	592	592	592	676.1

Tabla 15: Resultados para el programa 4_4_2011 con bloques de 100 iteraciones

A continuación, en el Gráfico 12, se muestra como dichos valores mínimos de cada bloque de iteraciones van variando a lo largo de la ejecución del programa, además de mostrar el valor exacto de algunos de los valores más relevantes.

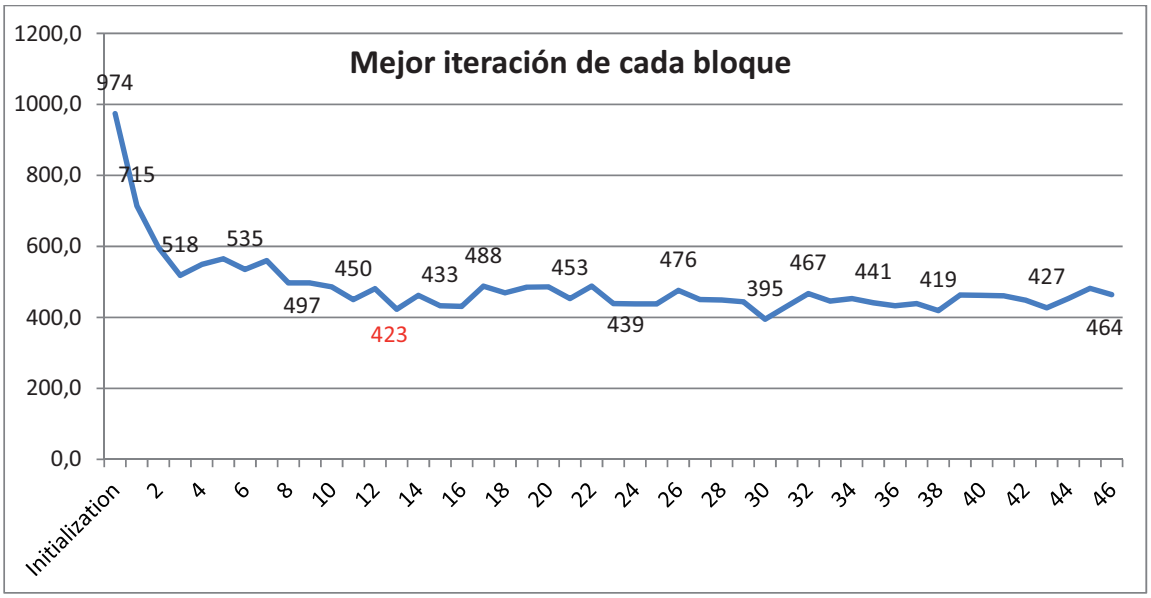

Gráfico 12: Valores mínimos de cada bloque de 100 iteraciones para el programa 4_4_2011

Se puede observar cómo, al igual que sucedía en la aplicación del presente programa con 11 grandes bloques de iteraciones en comparación con los resultados obtenidos con el programa a día 3_17_2011, en este caso, al aplicar 46 bloques de un reducido número de iteraciones también se consiguen unas mejoras realmente buenas con el respectivo programa aplicado el día 3_17_2011. Los valores de promedio se han reducido de cerca de 1200 a casi la mitad, alrededor de 600, los valores de derivación estándar, al igual que sucedía al comparar ambos programas aplicando grandes bloques de iteraciones, se han reducido enormemente, de un promedio de alrededor de 350 a un promedio de cerca de 45. Por otra parte, si bien el valor mínimo no se reduce enormemente, este sufre una ligera mejora, del valor obtenido por el programa anterior de 431, al valor del presente programa de 423.

Por tanto, tal y como se observó en el análisis de los resultados realizado para el programa del día 3_17_2011, se puede decir finalmente que, la aplicación de menos bloques de iteraciones con mayor cantidad de iteraciones en cada uno de ellos proporciona unos valores ligeramente mejores que el uso de más bloques de iteración pero con menos iteraciones en cada uno de ellos por lo que el futuro método de aplicación de los mismos será procediendo de la primera manera, es decir con grandes bloques de iteraciones.

Como ya se comentó anteriormente, los resultados que se han obtenido con esta modificación del programa proporcionan unos resultados realmente buenos y satisfactorios para el problema tratado, por lo cual, se va a considerar el mismo como el resultado final del presente proyecto, dejando para futuras ampliaciones del mismo, las posibles mejoras que pudieran realizarse a partir del esta última modificación del programa.

6.4. PROGRAMA DEFINIFTIVO

A continuación, a modo de conclusión de los anexos, se expone el programa desarrollado A día 4_4_2011, con el cual, tal y como se comentó anteriormente, se obtuvieron los mejores resultados. El lenguaje de programación ha sido Python.

```
# -*- coding: cp1252 -*-
import networkx as nx
import matplotlib.pyplot as plt
import math
import random as rd
import copy
# Put it only if IDLE is used and we want to depure
#import pdb
#pdb.set_trace()

# ---------------------------------------------------------------------------
# CREATION OF THE GRAPH
# ---------------------------------------------------------------------------
G = nx.Graph()

# Location of nodes with UTM coordinates (http://www.mundivideo.com/coordenadas.htm)
# Nodes are assigned in Google Maps (URI)

nbunch= { 0: (287092.96,4595420.82),\
          1: (287805.32,4596225.97),\
          2: (287617.43,4596298.07),\
          3: (287808.21,4596429.14),\
          4: (287774.67,4596249.21),\
          5: (287476.20,4596144.09),\
          6: (288124.75,4596174.25),\
          7: (288194.87,4596470.46),\
          8: (288070.32,4595985.15),\
          9: (287938.59,4596001.02),\
         10: (288287.88,4596451.17),\
         11: (288220.89,4596155.75),\
         12: (288334.74,4596436.46),\
         13: (288351.22,4596510.99),\
         14: (288537.94,4596459.06),\
         15: (288527.13,4596395.03),\
         16: (288448.74,4596411.67),\
         17: (288441.02,4596353.77),\
         18: (288505.24,4596343.89),\
         19: (288445.27,4596256.30),\
         20: (288507.58,4596269.70),\
         21: (288403.14,4596255.76),\
         22: (288372.67,4596285.21),\
         23: (288374.84,4596299.48),\
         24: (288375.87,4596328.90),\
         25: (288377.76,4596347.63),\
         26: (288263.95,4596350.52),\
         27: (288256.51,4596310.51),\
         28: (288251.92,4596290.53),\
         29: (288332.08,4596228.84),\
         30: (288303.97,4596139.43),\
         31: (288301.95,4596193.05),\
         32: (288242.73,4596054.21),\
         33: (288218.73,4595680.63),\
         34: (288274.12,4595674.56),\
```

35: (288363.97,4595663.93),\
36: (288380.17,4595880.04),\
37: (288343.82,4595577.84),\
38: (288217.36,4595434.64),\
39: (289125.94,4595462.04),\
40: (288136.96,4595445.89),\
41: (288271.24,4595650.53),\
42: (288193.95,4595659.47),\
43: (288163.56,4595557.68),\
44: (288203.72,4595550.72),\
45: (289341.93,4595767.44),\
46: (288127.99,4595413.59),\
47: (287850.38,4595470.42),\
48: (287783.51,4595515.28),\
49: (287911.52,4595461.07),\
50: (287917.42,4595493.90),\
51: (287932.62,4595584.58),\
52: (287974.62,4595640.46),\
53: (288024.37,4595564.43),\
54: (287998.80,4595473.62),\
55: (287846.54,4595507.54),\
56: (288742.19,4596348.95),\
57: (288640.45,4596073.33),\
58: (288386.00,4596161.58),\
59: (288377.45,4596123.49),\
60: (288503.63,4596101.01),\
61: (288517.55,4596134.05),\
62: (288413.76,4596156.32),\
63: (288726.84,4596056.02),\
64: (288641.03,4595942.07),\
65: (288645.57,4595865.93),\
66: (288640.37,4595796.96),\
67: (288583.25,4595789.63),\
68: (288554.01,4595938.29),\
69: (288505.37,4595977.72),\
70: (288509.35,4596036.50),\
71: (288553.67,4596020.53),\
72: (288425.97,4595883.14),\
73: (288620.60,4595478.60),\
74: (288413.86,4595424.43),\
75: (288471.23,4595408.86),\
76: (288590.20,4595367.81),\
77: (288211.76,4595397.69),\
78: (288293.28,4595379.29),\
79: (288438.14,4595548.74),\
80: (288452.68,4595357.61),\
81: (288425.72,4595338.73),\
82: (288570.60,4595297.93),\
83: (288498.75,4595857.34),\
84: (288755.87,4596051.61),\
85: (288762.04,4596125.11),\
86: (288813.95,4596334.40),\
87: (288787.66,4596141.36),\
88: (288832.45,4596124.38),\
89: (288829.92,4596037.89),\
90: (288953.28,4596041.39),\
91: (288930.34,4596117.07),\
92: (289001.62,4596054.31),\
93: (289028.74,4596156.20),\
94: (289109.80,4596153.39),\
95: (289138.71,4596282.01),\

96: (289289.24,4596235.16),\
97: (289255.46,4596137.02),\
98: (289084.72,4596107.23),\
99: (289064.26,4596067.60),\
100: (289160.48,4596060.78),\
101: (289036.50,4596052.85),\
102: (289213.43,4596048.12),\
103: (289372.57,4596109.93),\
104: (289473.41,4596077.98),\
105: (289494.71,4596149.26),\
106: (289181.60,4595971.26),\
107: (289339.35,4595925.21),\
108: (289441.71,4595899.44),\
109: (289156.12,4595897.00),\
110: (289049.69,4595906.44),\
111: (288952.41,4595906.07),\
112: (288940.30,4595766.29),\
113: (289121.21,4595746.22),\
114: (289120.06,4595675.24),\
115: (288938.13,4595677.89),\
116: (288863.01,4595683.65),\
117: (288871.63,4595764.18),\
118: (288856.05,4595599.95),\
119: (288629.80,4595587.35),\
120: (288708.84,4595578.25),\
121: (288772.69,4595584.05),\
122: (288668.15,4595444.20),\
123: (288800.13,4595388.11),\
124: (288781.48,4595301.98),\
125: (288771.77,4595247.36),\
126: (288745.43,4595326.70),\
127: (288822.10,4595236.45),\
128: (289008.68,4595192.65),\
129: (289143.58,4595161.81),\
130: (288983.61,4595346.96),\
131: (289012.22,4595482.37),\
132: (288960.67,4595497.21),\
133: (288969.96,4595617.51),\
134: (289121.36,4595473.84),\
135: (289263.11,4595829.64),\
136: (289127.40,4595803.60),\
137: (289203.26,4595883.17),\
138: (289186.14,4595835.00),\
139: (289223.83,4595711.99),\
140: (289242.47,4595783.79),\
141: (289297.41,4595762.07),\
142: (289271.49,4595701.60),\
143: (289347.52,4595804.40),\
144: (289355.16,4595682.27),\
145: (289406.36,4595669.99),\
146: (289348.09,4595623.46),\
147: (289521.26,4595584.07),\
148: (289680.89,4596106.83),\
149: (289634.86,4595958.04),\
150: (289603.15,4595853.61),\
151: (289589.84,4595812.44),\
152: (289571.92,4595750.95),\
153: (289472.99,4595418.90),\
154: (289385.97,4595108.84),\
155: (289247.77,4595135.55),\
156: (289179.62,4595548.93),\

```
    157: (289284.14,4595490.98),\
    158: (289275.50,4595441.11),\
    159: (289362.73,4595449.35),\
    160: (289329.14,4595329.31),\
    161: (289145.85,4595208.20),\
    162: (288958.62,4595275.35),\
    163: (288941.75,4595232.95),\
    164: (289127.50,4595209.18),\
    165: (289144.87,4595234.68),\
    166: (289114.52,4595240.01),\
    167: (288016.94,4596196.08),\
    168: (288079.07,4596451.07),\
    169: (289145.69,4595325.67),\
    170: (289119.77,4595556.90),\
    171: (289274.03,4595668.52),\
    172: (289349.58,4595645.76),\
    173: (289127.44,4595896.50),\
    174: (289683.29,4595790.60),\
    175: (289842.63,4595982.43),\
    176: (289762.04,4595969.55),\
    177: (289850.52,4595914.75),\
    178: (289774.48,4595857.51),\
    179: (290053.73,4596118.30),\
    180: (288810.17,4596036.69),\
    181: (288789.88,4595820.25),\
    182: (288739.47,4595796.72),\
    183: (288802.83,4595771.64),\
    184: (288804.66,4595802.82),\
    185: (288765.87,4595705.27),\
    186: (288747.39,4595579.02),\
    187: (288792.98,4595617.80),\
    188: (288813.22,4595589.98),\
    189: (288758.25,4595659.04),\
    190: (288776.29,4595647.29),\
    191: (288363.84,4595918.86),\
    192: (288544.18,4595893.46),\
    193: (288576.04,4595911.30),\
    194: (288664.68,4595831.03),\
    195: (288721.79,4596020.50),\
    196: (288285.64,4595073.14),\
    197: (288491.54,4595169.34),\
    198: (287947.01,4595127.31),\
    199: (289206.94,4595717.38),\
    200: (289500.81,4595774.03),\
    201: (288671.66,4595452.99)}

# Adding nodes to the graph
G.add_nodes_from(nbunch, attr_dict=None)

# Function to calculate distances between 2 points (UTM)
def distance(N1,N2):
   X1=nbunch[N1][0];
   X2=nbunch[N2][0];
   Y1=nbunch[N1][1];
   Y2=nbunch[N2][1];
   dist= math.sqrt(((X2-X1)**2)+((Y2-Y1)**2));
   return dist ;

# List of all the edges with their 'weight'
ebunch={0,1,{'mandatory': 1.0}),(0,47,{'mandatory': 0.0}),\
     (1,4,{'mandatory': 0.0}),(1,167,{'mandatory': 0.0}),\
```

(2,5,{'mandatory': 1.0}),(2,4,{'mandatory': 1.0}),\
(3,4,{'mandatory': 0.0}),\
(6,167,{'mandatory': 0.0}),(6,11,{'mandatory': 0.0}),(6,7,{'mandatory': 0.0}),(6,8,{'mandatory': 0.0}),\
(7,10,{'mandatory': 0.0}),\
(8,9,{'mandatory': 0.0}),\
(10,12,{'mandatory': 1.0}),(10,26,{'mandatory': 0.0}),\
(11,28,{'mandatory': 0.0}),(11,30,{'mandatory': 1.0}),(11,32,{'mandatory': 0.0}),\
(12,13,{'mandatory': 0.0}),(12,16,{'mandatory': 0.0}),\
(13,14,{'mandatory': 0.0}),\
(14,15,{'mandatory': 0.0}),\
(15,16,{'mandatory': 0.0}),(15,18,{'mandatory': 1.0}),(15,56,{'mandatory': 0.0}),\
(16,17,{'mandatory': 1.0}),\
(17,18,{'mandatory': 1.0}),(17,23,{'mandatory': 0.0}),\
(18,20,{'mandatory': 1.0}),\
(19,20,{'mandatory': 0.0}),(19,21,{'mandatory': 0.0}),(19,62,{'mandatory': 0.0}),\
(21,22,{'mandatory': 0.0}),(21,58,{'mandatory': 1.0}),\
(22,23,{'mandatory': 0.0}),\
(23,24,{'mandatory': 0.0}),\
(24,25,{'mandatory': 0.0}),(24,26,{'mandatory': 1.0}),\
(26,27,{'mandatory': 0.0}),\
(27,28,{'mandatory': 1.0}),\
(28,29,{'mandatory': 0.0}),\
(30,31,{'mandatory': 1.0}),(30,59,{'mandatory': 0.0}),\
(32,33,{'mandatory': 0.0}),(32,59,{'mandatory': 0.0}),\
(33,34,{'mandatory': 1.0}),\
(34,35,{'mandatory': 1.0}),(34,41,{'mandatory': 0.0}),\
(35,36,{'mandatory': 0.0}),(35,37,{'mandatory': 1.0}),\
(36,72,{'mandatory': 1.0}),(36,191,{'mandatory': 0.0}),\
(37,38,{'mandatory': 0.0}),(37,79,{'mandatory': 0.0}),\
(38,40,{'mandatory': 0.0}),\
(39,134,{'mandatory': 0.0}),(39,158,{'mandatory': 0.0}),(39,169,{'mandatory': 0.0}),\
(40,44,{'mandatory': 1.0}),(40,46,{'mandatory': 0.0}),\
(41,42,{'mandatory': 0.0}),(41,44,{'mandatory': 0.0}),\
(42,43,{'mandatory': 1.0}),\
(43,44,{'mandatory': 0.0}),\
(45,143,{'mandatory': 0.0}),\
(46,49,{'mandatory': 1.0}),(46,77,{'mandatory': 0.0}),\
(47,49,{'mandatory': 1.0}),(47,55,{'mandatory': 0.0}),\
(48,55,{'mandatory': 0.0}),\
(49,50,{'mandatory': 0.0}),\
(50,51,{'mandatory': 1.0}),(50,54,{'mandatory': 0.0}),\
(51,52,{'mandatory': 0.0}),(51,53,{'mandatory': 0.0}),\
(56,86,{'mandatory': 1.0}),(56,57,{'mandatory': 0.0}),\
(57,60,{'mandatory': 0.0}),(57,63,{'mandatory': 0.0}),(57,64,{'mandatory': 1.0}),\
(58,59,{'mandatory': 1.0}),(58,62,{'mandatory': 0.0}),\
(59,60,{'mandatory': 0.0}),\
(60,70,{'mandatory': 0.0}),\
(61,62,{'mandatory': 1.0}),\
(63,84,{'mandatory': 0.0}),(63,195,{'mandatory': 0.0}),\
(64,65,{'mandatory': 1.0}),\
(65,66,{'mandatory': 1.0}),(65,194,{'mandatory': 0.0}),\
(66,67,{'mandatory': 0.0}),(66,119,{'mandatory': 0.0}),(66,194,{'mandatory': 0.0}),\
(67,193,{'mandatory': 0.0}),\
(68,69,{'mandatory': 0.0}),(68,71,{'mandatory': 1.0}),(68,192,{'mandatory': 0.0}),(68,193,{'mandatory': 0.0}),\
(70,71,{'mandatory': 0.0}),\
(72,83,{'mandatory': 0.0}),\
(73,76,{'mandatory': 0.0}),(73,79,{'mandatory': 0.0}),\
(74,75,{'mandatory': 1.0}),(74,79,{'mandatory': 1.0}),\
(75,76,{'mandatory': 0.0}),(75,80,{'mandatory': 0.0}),\
(76,82,{'mandatory': 0.0}),\
(77,78,{'mandatory': 0.0}),(77,198,{'mandatory': 0.0}),\

(78,81,{'mandatory': 1.0}),(78,196,{'mandatory': 1.0}),\
(81,82,{'mandatory': 0.0}),(81,197,{'mandatory': 0.0}),\
(82,125,{'mandatory': 0.0}),\
(83,192,{'mandatory': 0.0}),\
(84,85,{'mandatory': 0.0}),(84,180,{'mandatory': 0.0}),\
(86,87,{'mandatory': 1.0}),(86,95,{'mandatory': 0.0}),\
(87,88,{'mandatory': 0.0}),\
(88,89,{'mandatory': 0.0}),\
(89,90,{'mandatory': 1.0}),(89,180,{'mandatory': 0.0}),\
(90,91,{'mandatory': 0.0}),(90,92,{'mandatory': 0.0}),\
(92,93,{'mandatory': 0.0}),(92,101,{'mandatory': 1.0}),\
(93,94,{'mandatory': 1.0}),\
(94,95,{'mandatory': 0.0}),(94,96,{'mandatory': 0.0}),(94,97,{'mandatory': 1.0}),(94,98,{'mandatory': 0.0}),\
(95,96,{'mandatory': 0.0}),\
(96,97,{'mandatory': 1.0}),(96,105,{'mandatory': 0.0}),\
(97,103,{'mandatory': 0.0}),(97,102,{'mandatory': 0.0}),\
(98,99,{'mandatory': 0.0}),(98,100,{'mandatory': 0.0}),(98,101,{'mandatory': 0.0}),\
(99,100,{'mandatory': 0.0}),(99,101,{'mandatory': 0.0}),\
(100,102,{'mandatory': 1.0}),\
(101,110,{'mandatory': 0.0}),\
(102,106,{'mandatory': 1.0}),\
(103,104,{'mandatory': 0.0}),(103,107,{'mandatory': 0.0}),\
(104,105,{'mandatory': 1.0}),(104,108,{'mandatory': 0.0}),\
(105,148,{'mandatory': 0.0}),\
(106,107,{'mandatory': 1.0}),(106,109,{'mandatory': 0.0}),\
(107,108,{'mandatory': 0.0}),\
(108,150,{'mandatory': 0.0}),\
(109,136,{'mandatory': 0.0}),(109,137,{'mandatory': 1.0}),(109,173,{'mandatory': 0.0}),\
(110,111,{'mandatory': 0.0}),(110,173,{'mandatory': 0.0}),\
(111,112,{'mandatory': 0.0}),\
(112,113,{'mandatory': 1.0}),(112,115,{'mandatory': 0.0}),(112,117,{'mandatory': 0.0}),\
(113,114,{'mandatory': 1.0}),(113,136,{'mandatory': 0.0}),(113,199,{'mandatory': 1.0}),\
(114,115,{'mandatory': 1.0}),(114,170,{'mandatory': 0.0}),\
(115,116,{'mandatory': 1.0}),\
(116,117,{'mandatory': 1.0}),(116,118,{'mandatory': 0.0}),\
(118,122,{'mandatory': 1.0}),(118,188,{'mandatory': 0.0}),\
(119,120,{'mandatory': 0.0}),\
(120,201,{'mandatory': 0.0}),(120,186,{'mandatory': 0.0}),\
(121,186,{'mandatory': 0.0}),(121,187,{'mandatory': 1.0}),(121,188,{'mandatory': 0.0}),(121,190,{'mandatory': 0.0}),\
(122,123,{'mandatory': 0.0}),(122,132,{'mandatory': 1.0}),\
(123,124,{'mandatory': 0.0}),(123,130,{'mandatory': 0.0}),\
(124,125,{'mandatory': 0.0}),(124,126,{'mandatory': 0.0}),\
(125,127,{'mandatory': 0.0}),\
(127,128,{'mandatory': 0.0}),\
(128,129,{'mandatory': 1.0}),\
(129,161,{'mandatory': 0.0}),(129,155,{'mandatory': 0.0}),\
(130,131,{'mandatory': 1.0}),(130,169,{'mandatory': 0.0}),\
(131,132,{'mandatory': 0.0}),(131,134,{'mandatory': 0.0}),\
(132,133,{'mandatory': 1.0}),\
(134,169,{'mandatory': 0.0}),(134,158,{'mandatory': 0.0}),(134,170,{'mandatory': 1.0}),\
(135,137,{'mandatory': 0.0}),(135,140,{'mandatory': 1.0}),(135,143,{'mandatory': 0.0}),\
(136,173,{'mandatory': 1.0}),\
(137,138,{'mandatory': 0.0}),\
(139,140,{'mandatory': 0.0}),(139,142,{'mandatory': 0.0}),(139,199,{'mandatory': 0.0}),\
(140,141,{'mandatory': 0.0}),\
(141,142,{'mandatory': 1.0}),\
(142,144,{'mandatory': 0.0}),(142,171,{'mandatory': 0.0}),\
(143,144,{'mandatory': 0.0}),\
(144,145,{'mandatory': 0.0}),(144,172,{'mandatory': 1.0}),\
(146,172,{'mandatory': 0.0}),(146,147,{'mandatory': 1.0}),\
(147,152,{'mandatory': 1.0}),(147,153,{'mandatory': 0.0}),\

```
    (148,149,{'mandatory': 0.0}),(148,179,{'mandatory': 1.0}),\
    (149,150,{'mandatory': 0.0}),(149,176,{'mandatory': 0.0}),\
    (150,151,{'mandatory': 1.0}),\
    (151,152,{'mandatory': 0.0}),(151,200,{'mandatory': 0.0}),\
    (152,174,{'mandatory': 0.0}),\
    (153,154,{'mandatory': 0.0}),(153,159,{'mandatory': 0.0}),\
    (154,155,{'mandatory': 1.0}),\
    (156,170,{'mandatory': 0.0}),(156,157,{'mandatory': 1.0}),(156,199,{'mandatory': 0.0}),\
    (157,158,{'mandatory': 0.0}),\
    (158,159,{'mandatory': 0.0}),\
    (159,160,{'mandatory': 1.0}),\
    (161,164,{'mandatory': 0.0}),(161,165,{'mandatory': 0.0}),\
    (162,163,{'mandatory': 0.0}),(162,164,{'mandatory': 0.0}),\
    (163,164,{'mandatory': 0.0}),\
    (165,166,{'mandatory': 0.0}),(165,169,{'mandatory': 0.0}),\
    (167,168,{'mandatory': 1.0}),\
    (171,172,{'mandatory': 0.0}),\
    (175,176,{'mandatory': 1.0}),(175,177,{'mandatory': 0.0}),\
    (176,178,{'mandatory': 1.0}),\
    (180,181,{'mandatory': 0.0}),\
    (181,182,{'mandatory': 1.0}),(181,184,{'mandatory': 0.0}),\
    (183,184,{'mandatory': 0.0}),\
    (185,189,{'mandatory': 1.0}),\
    (186,189,{'mandatory': 0.0}),\
    (187,190,{'mandatory': 1.0}),(187,188,{'mandatory': 0.0}),\
    (189,190,{'mandatory': 0.0}),\
    (192,193,{'mandatory': 0.0})

# Adding edges to the graph
edge_data = {'weight': 1.0, 'pheromone':0.0, 'probability': 0.0}

G.add_edges_from(ebunch,attr_dict=edge_data)

mandatory_edges=[]
for u,v in G.edges_iter():
      if (G.get_edge_data(u,v, default=0.0)['mandatory'] == 1.0):
         mandatory_edges.append((u,v))

print 'Number of edges =',len(ebunch)
print 'Number of mandatory edges =', len(mandatory_edges)

# ----------------------------------------------------------------------------------------
# CREATION OF THE DIFFERENT RANDOM PATHS GIVEN FOR THE FIRST 'ANTS' (not using 'pheromone' values)
# ----------------------------------------------------------------------------------------

# An empty dictionary is created to allow memorize all the paths produced by the first random 'ants'
dict_of_paths={}

# An empty list is created in which the edges traversed will be added
visited=[]

# Added a flag which will advise when all the mandatory edges have been already recovered
continue_flag = 0.0

print 'Calculating the block of initialization....'

iteration=0
while (iteration < 500):

   #print "Iteration number =", iteration
```

```
# Obtaining the minimum weight of all the edges(Valid in directed and undirected graphs)
start_node = current = 1  # this is the origin node

# Checking if the mandatory edges have been already recovered
for u,v in G.edges_iter():
   if (G.get_edge_data(u,v, default=0.0)['weight'] == \
      G.get_edge_data(u,v, default=0.0)['mandatory']):
      continue_flag = 1.0

#print "INITIAL INFORMATION"
#print "Start node = ", start_node
#print "Minimum weight of all the edges = ", min_total_weight
#print "-----------------------------------------------"

while (continue_flag == 1.0):
   #print "Not all the edges of the graph have been recovered"
   list_all_neighbors = G.neighbors(current)
   num_neighbors = len(list_all_neighbors)

   #Information regarding the neighbors
   #print "Current node = ", current
   #print "List of all neighbors = ", list_all_neighbors
   #print "Weight of the neighbors = ", [G.get_edge_data(current,v, default=1000)['weight'] \
   #                              for v in (list_all_neighbors)]

   # Information about the value of S (sum of the weight of all the neighbors)
   sum_weight_neighbors = 0.0
   for X in (list_all_neighbors):
      sum_weight_neighbors = sum_weight_neighbors + \
      G.get_edge_data(current,X,default=1000)['weight']
   #print "Total sum of the weight of all the neighbors (S) = ", sum_weight_neighbors


   #Information about the calculus of the probability in each edge
   for X in (list_all_neighbors):
      #print "Calculating probability of edge...", [current, X]
      #print "Weight of this edge = " , G.get_edge_data(current,X, default=1000)['weight']
      if num_neighbors==1:
         G[current][X]['probability'] = 1.0
         #print 'There are only one neighbor'
      elif num_neighbors!=1:
          a = (1.0/(num_neighbors - 1.0))
          b = sum_weight_neighbors - G.get_edge_data(current,X, default=1000)['weight']
          c = (b/sum_weight_neighbors)
          G[current][X]['probability'] = a*c
          #print "a",a
          #print "b",b
          #print "c",c
      #print "Probability of the edge = ", G[current][X]['probability']

   #Information about the procedure of selecting the next edge
   random = rd.random()
   #print "Selecting the next edge...."
   #print "random =", random
   total_probability = 0.0
   flag = 0
   for X in (list_all_neighbors):
      total_probability = total_probability + \
      G.get_edge_data(current,X, default=1000)['probability']
      #print "X", X
      #print "Total probability for the neighbor", [X],"is...", total_probability
```

```
            if random < total_probability and flag!=1:
                next_node = X
                #print "Next node will be:" , next_node
                flag = 1

        # Adding the new edge traversed to our list of edges traversed
        visited = visited + [(current, next_node)]
        # Updating edges' information
        G[current][next_node]['weight'] = G[current][next_node]['weight'] * 10;

        #Information about the selected edge
        #print "The edge selected is..", [current, next_node]

        #Resetting the probability of the edges
        for X in (list_all_neighbors):
            G[current][X]['probability'] = 0.0

        #Assigning next node as a current to start the procedure again
        current = next_node

        # Stop the procedure until was checked if the mandatory edges have been recovered
        continue_flag= 0.0

        # Checking if the mandatory edges have been already recovered
        for u,v in G.edges_iter():
            if (G.get_edge_data(u,v, default=0.0)['weight'] ==\
                G.get_edge_data(u,v, default=0.0)['mandatory']):
                continue_flag=1.0

        #print "-----------------------------------------"

    # Once all edges have been traversed, the shortest path between the current node and
    # the 'start_node' is calculated taking into account the weight of the edges (if weighted=True)
    # and it is represented in BLUE DOTTED LINE. In the case that all the edges had the same 'weight'
    # it could be more than one shortest path so only one will be represented. In the case that all
    # the edges had the same weight or the parameter weighted=False, the shortest path will be
    # calculated assigning the same 'weight' to every of them. This is the case of the most common
    # programs we are developing because in our case the parameter 'weight' is increasing as much
    # times as one edge is recovered, so it is no sense to use the algorithm 'shortest_path' with the
    # parameter weighted=True, because this parameter doesn't represent in our case distances,time...

    #print "Last node visited before 'shortest path' = ", current
    nodes_shortest = nx.shortest_path(G, source=current, target=start_node, weighted=False)
    I = G.subgraph(nodes_shortest)

    # Added the edges of the shortest path
    visited = visited + I.edges()

    #Summary of the iteration
    #print "List of edges visited = ",visited
    #print len(visited)

    #Saving the list into the correspondent place in the dictionary
    dict_of_paths[iteration] = visited

    # Reinitializing parameters for the next iteration
    iteration = iteration + 1

    #Reseting the list of visited edges
    visited = []
```

```
    # Reseting the weight of all edges in order to be able to generate another iteration
    # (the value 'weight' indicates in the program if an edge has or hasn't been already recovered
    # and we use that to know when the iteration finishes)
    for u,v in G.edges_iter():
        G[u][v]['weight']= 1.0

    #print "Final of the present iteration"
    #print "&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"



# ----------------------------------------------------------------------------------------------------
# FINAL RESUME OF THE FIRST BLOCK OF ITERATIONS
# ----------------------------------------------------------------------------------------------------

print "--------------------------------------------------------------------------------"
print "        FINAL RESUME OF THE BLOCK OF INITIALIZATION                    "
print "--------------------------------------------------------------------------------"

print "Length of the paths:"
for X in dict_of_paths.keys():
    print len(dict_of_paths[X])


print 'Calculating and updating the pheromone values for the next block of iterations'

# ----------------------------------------------------------------------------------------------------
# ONCE THE FIRST ANTS HAVE FINISHED ALL THE ITERATIONS WE CALCULATE A 'PHEROMONE' VALUE DEPENDING TO
# THE QUALITY OF THE SOLUTION OBTAINED AND THIS VALUE IS STORED IN EACH EDGE OF THE GRAPH
# ----------------------------------------------------------------------------------------------------

# The program has finished to develop the 'random' iterations and all the data have been saved in a
# dictionary with the name of dict_of_paths on which the keys are the iteration number and the items
# are the list of the edges recovered in the correspondent iteration.

#Creation of a list with the distances of all the iterations
list_of_lengths = []
for X in dict_of_paths.keys():
    list_of_lengths.append( len(dict_of_paths[X]) );

# Information of the distances taken in the different iterations and the minimum of all of them
# (shortest_path_length) as a float number in order to be able to do operations later.
shortest_path_length = float(min(list_of_lengths))
#print 'List of the lengths of the different iterations = ', list_of_lengths
#print 'Shortest length of the different iterations', shortest_path_length

# Iteration over all the paths created (all the iterations)
for X in dict_of_paths.keys():
    # Getting the length of the iteration which is being studied
    current_path_length = float(len(dict_of_paths[X]))

    # Calculating the value ('pheromone') to update in all the edges of the current path
    pheromone_value_to_update = \
    ((math.e)**(-4.0*(1.0 - (current_path_length / shortest_path_length))**2))

    # Information about the values to update in the current path
    #print 'Path n :', X
    #print 'Length of the current path', current_path_length
    #print 'Pheromone value to update for all the edges of this path = ', pheromone_value_to_update
```

```
  # In the path who is being studied actualization of the 'pheromone value for all the edges
  # this path contains
  for u,v in dict_of_paths[X]:
    G[u][v]['pheromone']= G[u][v]['pheromone'] + pheromone_value_to_update

# Checking if the results are coherent

print 'Edges...'
for u,v in G.edges_iter():
  print [u,v]

print 'Pheromone values...'
for u,v in G.edges_iter():
  print G.get_edge_data(u,v, default=0)['pheromone']

# Introducing another constrain to the program
best_so_far = shortest_path_length
limit_of_length= 1.5 * best_so_far
print 'Best so far:', best_so_far
print 'Limit of length:', limit_of_length

#print
'&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&'

number_of_blocks = 0
while (number_of_blocks < 12):

  # -----------------------------------------------------------------------------------------
  # CREATION OF THE PATHS TAKEN INTO ACCOUNT ALSO THE 'PHEROMONE' VALUE OF ALL THE EDGES AND USING
  # THE VALUE 'WEIGHT' TO KNOW IF AN EDGE HAS OR HASN'T BEEN RECOVERED AND TO KNOW IN THIS WAY
  # WHEN ALL THE EDGES HAVE BEEN RECOVERED.
  # ALSO THE PARAMETER 'PROBABILITY' WILL BE RE-USED TO SELECT WITH A PROBABILISTIC METHOD WITH
  # EDGE TO CHOOSE.
  # -----------------------------------------------------------------------------------------

  # Reseting the values of the dictionary to memorize the next block of iterations
  dict_of_paths={}

  # An empty list is created in which the edges traversed will be added
  visited=[]

  # Added a flag which will advise when all the mandatory edges have been already recovered
  continue_flag = 0.0

  print 'Calculating the next block of iterations...'

  iteration=0
  while (iteration < 1000):
    print 'Iteration n =', iteration

    start_node = current = 1  # this is the origin node

    # Checking if the mandatory edges have been already recovered
    for u,v in G.edges_iter():
      if (G.get_edge_data(u,v, default=0.0)['weight'] == \
        G.get_edge_data(u,v, default=0.0)['mandatory']):
        continue_flag = 1.0

    #print "INITIAL INFORMATION"
    #print "Start node = ", start_node
    #print "Minimum weight of all the edges = ", min_total_weight
```

```
#print "-----------------------------------------------"

while (continue_flag == 1.0):

  #print "Not all the edges of the graph have been recovered"
  list_all_neighbors = G.neighbors(current)
  num_neighbors = len(list_all_neighbors)
  min_neighbors_weight= min(G.get_edge_data(current,v, default=1000)['weight'] \
                for v in (list_all_neighbors))

  #Information regarding the neighbors
  #print "Current node = ", current
  #print "List of all neighbors = ", list_all_neighbors

  # Selection of the edge in case that there exists some 'virgin'edge
  if min_neighbors_weight == 1.0:
    list_neighbors_to_take=[]
    for X in (list_all_neighbors):
      if (G.get_edge_data(current,X, default=1000)['weight'] == 1.0):
          list_neighbors_to_take.append(X)
    #print 'List of neighbors to consider...', list_neighbors_to_take

    #In case there were only ONE 'virgin' edge the probability will be 100%
    if len(list_neighbors_to_take)==1:
      for X in list_neighbors_to_take:
        G[current][X]['probability']= 1.0
        #print 'Probability edge',[current,X],'=',G[current][X]['probability']

    #In case there were more than one 'virgin' edge we have to calculate the probability
    #taking into account the 'pheromone' value
    else:

      #Creation of a list with all the useful 'pheromone' values in this selection and
      #creation of a dictionary with 'keys' the number of the node considered and the
      #'values' the 'pheromone' value.
      list_pheromones=[]
      pheromone_values={}
      for X in (list_neighbors_to_take):
        pheromone_values[X]= (G.get_edge_data(current,X, default=1000)['pheromone'])
        list_pheromones.append(G.get_edge_data(current,X, default=1000)['pheromone'])

      # Procedure to calculate the 'probability' (differs to the first 'ants'
      # procedure of selection)
      minimum_pheromone = min(list_pheromones)
      maximum_pheromone = max(list_pheromones)

      # Creation to a dictionary with the 'keys' the number of the nodes and the
      # 'values' the 'new_weight' used to calculate the value of probability.
      new_weights={}
      for X in pheromone_values.keys():
        current_edge_pheromone = pheromone_values[X]
        new_weights[X]=(maximum_pheromone-minimum_pheromone)+\
        (current_edge_pheromone-minimum_pheromone)
        # print 'Node', X,'New_weight', new_weights[X]

      sum_new_weights_squared = 0.0
      for X in new_weights.keys():
        sum_new_weights_squared = sum_new_weights_squared + ((new_weights[X])**2)
      #print 'Denominator..', sum_new_weights_squared

      for X in new_weights.keys():
```

```
            G[current][X]['probability']=(((new_weights[X])**2)/sum_new_weights_squared)
            #print 'Probability edge',[current,X],'=',G[current][X]['probability']

        #Information about the procedure of selecting the next edge
        random = rd.random()
        #print "Selecting the next edge...."
        #print "random =", random
        total_probability = 0.0
        flag = 0
        for X in (list_all_neighbors):
            total_probability = total_probability + \
            G.get_edge_data(current,X, default=1000)['probability']
            #print "X", X
            #print "Total probability for the neighbor", [X],"is...", total_probability

            if random < total_probability and flag!=1:
                next_node = X
                #print "Next node will be:" , next_node
                flag = 1.0

    # Selection of the edge in case that there not exists any 'virgin'edges
    else:
        list_neighbors_to_take=[]
        for X in (list_all_neighbors):
            if (G.get_edge_data(current,X, default=1000)['weight'] != 1.0):
                list_neighbors_to_take.append(X)
        #print 'List of neighbors to consider...', list_neighbors_to_take

        num_neighbors = len(list_neighbors_to_take)

        # Information about the value of S (sum of the weight of all the neighbors)
        sum_weight_neighbors = 0.0
        for X in (list_neighbors_to_take):
            sum_weight_neighbors = sum_weight_neighbors + \
            G.get_edge_data(current,X,default=1000)['weight']
            #print "Total sum of the weight of all the neighbors(S)= ", sum_weight_neighbors

        #Information about the calculus of the probability in each edge
        for X in (list_neighbors_to_take):
        #print "Calculating probability of edge...", [current, X]
        #print "Weight of this edge = " , G.get_edge_data(current,X, default=1000)['weight']
            if num_neighbors==1:
                G[current][X]['probability'] = 1.0
                #print 'There are only one neighbor'
            else:
                a = (1.0/(num_neighbors - 1.0))
                b = sum_weight_neighbors- G.get_edge_data(current,X, default=1000)['weight']
                c = (b/sum_weight_neighbors)
                G[current][X]['probability'] = a*c
                #print "a",a
                #print "b",b
                #print "c",c
                #print 'Probability edge',[current,X],'=',G[current][X]['probability']

        #Information about the procedure of selecting the next edge
        random = rd.random()
        #print "Selecting the next edge...."
        #print "random =", random
        total_probability = 0.0
        flag = 0
        for X in (list_neighbors_to_take):
```

```
            total_probability = total_probability + \
            G.get_edge_data(current,X, default=1000)['probability']
            #print "X", X
            #print "Total probability for the neighbor", [X],"is...", total_probability

            if random < total_probability and flag!=1:
                next_node = X
                #print "Next node will be:" , next_node
                flag = 1


        # Adding the new edge traversed to our list of edges traversed
        visited = visited + [(current, next_node)]

        # Updating edges' information
        G[current][next_node]['weight'] = G[current][next_node]['weight'] * 10;
        #print "current and next", current, next_node

        #Information about the selected edge
        #print "The edge selected is..", [current, next_node]

        #Reseting the probability of the edges
        for X in (list_all_neighbors):
            G[current][X]['probability'] = 0.0

        #Assigning next node as a current to start the procedure again
        current = next_node

        # Stop the procedure until was checked if the mandatory edges have been recovered
        continue_flag= 0.0

         # Checking if the mandatory edges have been already recovered
        for u,v in G.edges_iter():
            if (G.get_edge_data(u,v, default=0.0)['weight'] ==\
                G.get_edge_data(u,v, default=0.0)['mandatory']):
                continue_flag=1.0

        #print "-----------------------------------------"

    #Once all edges have been traversed, the shortest path between the current node and
    #the 'start_node' is calculated taking into account the weight of the edges(weighted=True)
    #and it is represented in BLUE DOTTED LINE. In the case that all the edges had the same 'weight'
    #it could be more than one shortest path so only one will be represented. In the case that
    #all the edges had the same weight or the parameter weighted=False, the shortest path will be
    #calculated assigning the same 'weight' to every of them. This is the case of the most common
    #programs we are developing because in our case the parameter 'weight' is increasing as much
    #times as one edge is recovered, so it is no sense to use the algorithm 'shortest_path' with
    #parameter weighted=True,because this parameter doesn't represent in our case distances,time...

    #print "Last node visited before 'shortest path' = ", current
    nodes_shortest = nx.shortest_path(G, source=current, target=start_node, weighted=False)
    I = G.subgraph(nodes_shortest)

    # Added the edges of the shortest path
    visited = visited + I.edges()

    #Summary of the iteration
    #print "List of edges visited = ",visited
    #print len(visited)

    #Only saving the path in the case that the length is less or equal to the limit_of_length
```

```
    #If it is longer than the limit, the path is not saved and the number of iteration remains
    #constants for what it is repeated again.
    if (len(visited) <= limit_of_length):
        #Saving the list into the correspondent place in the dictionary
        dict_of_paths[iteration] = visited
        # Reinitializing parameters for the next iteration
        iteration = iteration + 1

    #Reseting the list of visited edges
    visited = []

    #Reseting the weight of all edges
    for u,v in G.edges_iter():
        G[u][v]['weight']= 1.0

    #print "Final of the present iteration"
    #print "&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"

# -------------------------------------------------------------------------------------------
# FINAL RESUME OF THE LAST BLOCK
# -------------------------------------------------------------------------------------------

print "--------------------------------------------------------------------------------"
print "     FINAL RESUME OF THE BLOCK OF ITERATIONS                                     "
print "--------------------------------------------------------------------------------"

print "Length of the paths:"
for X in dict_of_paths.keys():
    print len(dict_of_paths[X])


print 'Calculating and updating the pheromone values for the next block of iterations'

# Removing all the 'pheromone' values of the previous block
for u,v in G.edges_iter():
    G[u][v]['pheromone']= 0.0

#-------------------------------------------------------------------------------------------
#ONCE THE FIRST ANTS HAVE FINISHES ALL THE ITERATIONS WE CALCULATE A 'PHEROMONE' VALUE DEPENDING
#TO THE QUALITY OF THE SOLUTION OBTAINED AND THIS VALUE IS STORED IN EACH EDGE OF THE GRAPH
#-------------------------------------------------------------------------------------------

#The program has finished to develop the 'random' iterations and all the data have been saved in
#a dictionary with the name of dict_of_paths on which the keys are the iteration number and the
#items are the list of the edges recovered in the correspondent iteration.

#Creation of a list with the distances of all the iterations
list_of_lengths = []
for X in dict_of_paths.keys():
    list_of_lengths.append( len(dict_of_paths[X]) );

# Information of the distances taken in the different iterations and the minimum of all of them
# (shortest_path_length) as a float number in order to be able to do operations later.
shortest_path_length = float(min(list_of_lengths))
#print 'List of the lengths of the different iterations = ', list_of_lengths
#print 'Shortest length of the different iterations', shortest_path_length

# Iteration over all the paths created (all the iterations)
for X in dict_of_paths.keys():
    # Getting the length of the iteration which is being studied
    current_path_length = float(len(dict_of_paths[X]))
```

```
    # Calculating the value ('pheromone') to update in all the edges of the current path
    pheromone_value_to_update = \
    ((math.e)**(-4.0*(1.0 - (current_path_length / shortest_path_length))**2))

    # Information about the values to update in the current path
    #print 'Path n :', X
    #print 'Length of the current path', current_path_length
    #print 'Pheromone value to update for all the edges of this path=',pheromone_value_to_update

    # In the path who is being studied actualization of the 'pheromone value for all the edges
    # this path contains
    for u,v in dict_of_paths[X]:
      G[u][v]['pheromone']= G[u][v]['pheromone'] + pheromone_value_to_update

  # Checking if the results are coherent
   #print 'Edges...'
  #for u,v in G.edges_iter():
  #    print [u,v]

  print 'Pheromone values...'
  for u,v in G.edges_iter():
    print G.get_edge_data(u,v, default=0)['pheromone']

  #print '&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&'
   # Introducing another constrain to the program
  if  shortest_path_length < best_so_far:
    best_so_far = shortest_path_length

  limit_of_length= 1.5 * best_so_far
  print 'Best so far:', best_so_far
  print 'Limit of length for the next block:', limit_of_length

  # Increasing the counter of number of blocks
  number_of_blocks = number_of_blocks + 1

# ---------------------------------------------------------------------------
# ORDERS TO REPRESENT A GRAPH
# ---------------------------------------------------------------------------
nx.draw(G, pos=nbunch, with_labels=True)

nx.draw_networkx_edges(G, pos=nbunch, edgelist=None, width=1.0, edge_color='k', style='solid',\
                edge_cmap=None, edge_vmin=None, edge_vmax=None, ax=None, arrows=True)

# Mandatory edges
nx.draw_networkx_edges(G,pos=nbunch,edgelist=mandatory_edges,width=3.0,edge_color='r',style='solid',\
                edge_cmap=None, edge_vmin=None, edge_vmax=None, ax=None, arrows=True)

nx.draw_networkx_edges(I, pos=nbunch, width=5.0, edge_color='b', style='dotted', arrows=True)

# ---------------------------------------------------------------------------
# MATPLOTLIB ORDERS
# ---------------------------------------------------------------------------

plt.suptitle('Rural Chinese Postman Problem')
plt.text(1.5, -1.2, "Shortest path between last node and start node is the dotted blue line",\
      horizontalalignment='center', verticalalignment='center')
plt.savefig("Graph.png")

plt.show()
```