8. Conclusiones

El algoritmo implementado en este proyecto realiza una optimización del Problema de Steiner. El resultado del mismo, converge para la mayoría de los problemas optimizados, independientemente de que este resultado sea todo lo bueno que cabría esperar.

Si bien encontramos que en la mayoría de los problemas *"pequeños"* , es decir, los *steinb*, el error de la solución es nulo. Para problemas más complejos encontramos que el error crece. El algoritmo, sin embargo, es bastante eficaz en la mayoría de ellos ya que el número de iteraciones para llegar a la solución óptima, o cercana a la óptima, es muy bajo.

El algoritmo de Kruskal proporciona, en estos problemas, una solución inicial muy buena y da como resultado un valor próximo a la solución óptima del problema, con lo cual, el algoritmo desarrollado e implementado en este proyecto sólo tiene que iterar unas cuantas veces (en problemas no excesivamente complejos) para encontrar una solución aceptable.

Todo ello repercute directamente en el tiempo de ejecución, que es bastante bajo para problemas pequeños. Todas las ejecuciones del algoritmo se han llevado a cabo en una máquina no muy potente, un AMD K6-2 350Mhz con 256Mb de RAM, por lo que no se puede esperar un tiempo de ejecución excesivamente bajo, el hardware ha sido una limitación.

Es importante recalcar que debido a que este algoritmo es una heurística que trata de encontrar la mejor solución posible, y puesto que éste ha sido un Proyecto de investigación en el que se ha intentado depurar al máximo el resultado, se han realizado multitud de planteamientos en los que se han variado distintos parámetros.

Destacar que la optimización del problema de Steiner es una línea de investigación abierta en el grupo de Ingeniería de Organización y que, por ello, se han ido realizando modificaciones en el planteamiento a lo largo del proyecto.

Como ejemplo, a lo largo del desarrollo se ha variado la forma de calcular la solución admisible en cada iteración, la depuración de esa solución, la elección del nodo origen, la forma de recalcular el flujo que circula por cada arco, la función de integridad utilizada, etc.

El planteamiento inicial, a partir de cuyas deficiencias se han extraído los demás, consistía en ejecutar tantas veces el algoritmo de Dijkstra por iteración como número de nodos terminales existieran.

Esto se hacía porque es evidente que, si en cada iteración tomamos siempre el mismo origen y ejecutamos el algoritmo de la ruta mínima, estaremos beneficiando siempre a los arcos que estén directamente unidos a este nodo. Esto es así por el método original para calcular el peso de los arcos en la siguiente iteración.

Este método calculaba el peso a partir del flujo que circulase por cada arco, a mayor flujo más disminución del peso del arco en cada iteración, hasta ahí bien hasta que se plantea la forma en calcular ese flujo. Si se toma siempre el mismo nodo como destino de información, todo el tráfico generado en el resto de nodos terminales acabará pasando por los arcos de ese nodo que formen parte de la solución y con ello esos arcos se verán beneficiados.

En resumen, la solución depende del nodo escogido como origen a la hora de recalcular el flujo, porque los arcos de ese nodo pertenecientes a la solución en cada iteración soportarán todo el tráfico generado por el resto de nodos terminales hacia el origen y, por ello, se incluyen rápidamente en la

solución.

Esta conclusión se ha extraído de la propia optimización, ya que los arcos que tenían como origen o como destino el nodo origen veían disminuido considerablemente su peso en unas cuantas iteraciones.

De ahí, que se hayan estudiado otras variaciones, otros planteamientos, del algoritmo para ver si podíamos encontrar una forma más "justa" de recalcular el coste de los arcos. La solución a esto ha sido la de cambiar el *origen* en cada iteración o la de calcular en cada iteración el flujo por cada arco tomando como *origen* cada uno de los nodos terminales y luego superponer las soluciones.

Por un lado, existe esta deficiencia en nuestro algoritmo, y por el otro, cabe plantearse la conveniencia de utilizar el algoritmo de la ruta mínima en cada iteración. Si en cada iteración se calcula una solución a partir de Dijkstra para uno solo de los nodos terminales, encontraremos que esa solución en cada iteración depende de ese nodo que se escoja, por ello el planteamiento inicial ejecuta tantas veces Dijkstra en cada iteración como número de nodos terminales haya. Esto es totalmente inviable cuando el número de terminales crece, puesto que el algoritmo de Dijkstra consume un tiempo considerable, depende de n (nº de nodos, tanto Terminales como Steiner) como $O(n^2)$, y eso hace que para problemas con número de nodos elevado el tiempo que emplea el algoritmo se dispare, como se puede observar en las tablas de resultados para los planteamientos Primero y Segundo.

La única diferencia entre el Primer y el Segundo planteamiento es la función de integridad utilizada. En ambos, el tiempo de optimización es alto para problemas complejos, ya que en ambos se ejecuta el algoritmo de Dijkstra un número elevado de veces en cada iteración. Concretamente, tantas veces como número de Nodos Terminales tengamos. Por ello, encontramos que problemas con grafos de la misma magnitud se resuelven con tiempos de ejecución distintos, porque difieren en el número de Terminales.

Sin embargo, el tiempo de ejecución es menor cuando utilizamos la función de integridad Nº2. Ésta función hace que el algoritmo converja más rápidamente, puesto que la disminución del coste del arco en cada iteración es más severa si se utiliza esta función.

En el tercer procedimiento mejora tanto el tiempo de optimización como el resultado final. Es obvio que el tiempo de optimización disminuye debido a que solo se ejecuta una vez el algoritmo de la ruta mínima en cada iteración, eso es lo que ralentiza al primer y segundo planteamiento. Además, el Coste de la Solución es más bajo, aunque para problemas del tipo *steinc*, como cabe esperar por su complejidad, la solución no es excesivamente buena.

De la evolución del Coste de la Solución en cada iteración extraemos que en la mayoría de los problemas la solución final ofrecida por el algoritmo es inferior al Coste de la Solución Inicial que nos proporciona *Kruskal*. Sea el resultado mejor o peor, el algoritmo acaba convergiendo.

También se han realizado pruebas sin cambiar el Terminal el origen en todo el problema. Pero esto no lleva a buenas soluciones puesto que los arcos que conectan a ese nodo con el resto se ven beneficiados respecto al resto del grafo. Por ello, cambiar en cada iteración el Nodo Terminal que se utiliza como nodo *origen* se hace imprescindible.

La validez del sistema original para la resolución del problema de Steiner se comprobó mediante la utilización de Matlab en su optimización discreta. Sin embargo, Matlab tampoco proporciona resultados eficientes debido a las limitaciones que ofrece en la resolución de problemas de tamaño considerable.

Cabe concluir, por tanto, que el algoritmo presentado en este proyecto converge, a una solución

bastante buena para problemas no muy complejos. Si bien, en algunos problemas la solución es la óptima, para otros (los más complejos) encontramos desviaciones de hasta un 20%. Con un tiempo de optimización no excesivamente alto (dado la máquina en la que se ha ejecutado), al menos, la fiabilidad del método ha sido acotada puesto que converge, pero con un margen de error del 0-15% en la mayoría de los casos.