Práctica 3.

Metaheurísticas Basadas en Trayectorias

Metaheurísticas – Grado de Ingeniería Informática

Universidad de Córdoba

2015 / 2016

Grupo: NPGroup

Miembros:

- Daniel Carmona Martínez
- Cristian García Jiménez
- Antonio Jesús Jiménez Urbano
- Guillermo Sugráñez Pérez

Introducción

En esta práctica se ha abordado nuevamente la resolución de los problemas de la mochila y el viajante de comercio (vistos anteriormente en la práctica 1 y 2), pero siguiendo dos nuevas metaheurísticas basadas en trayectorias (Enfriamiento Simulado e Iterated Greedy) con el objetivo de comparar diversas medidas de rendimiento (fitness, tiempo) en función de las iteraciones realizadas. A la hora de realizar optimizaciones para los tiempos de ejecución se introdujo en las distintas estrategias la posibilidad de elegir el número de soluciones a evaluar.

Este documento consta de tres secciones diferentes. Una primera sección donde se detallan los aspectos de la codificación de cada nueva metaheurística, una segunda donde se evalúan las pruebas realizadas, una tercera donde se listan las conclusiones obtenidas a partir de dichas pruebas y una cuarta donde se continua con el estudio de nuestro problema (Máxima Diversidad) en base a los estudios realizados en esta práctica.

Capítulo 1 - Codificación

Enfriamiento Simulado (simulated annealing)

std::vector<double> EnfriamientoSimulado()

Este algoritmo se basa en el concepto de vecindario (si encuentra una solución vecina que sea mejor la acepta), pero puede evitar el problema de estancamiento en óptimos locales. Esto lo consigue añadiendo una nueva característica, la temperatura, que permite aceptar soluciones peores, permitiendo abandonar el óptimo local y dirigirse en las sucesivas iteraciones siguientes hacia el óptimo global. Esto lo consigue reduciendo progresivamente la temperatura. Cuando la temperatura es alta (principio del problema) será más probable aceptar soluciones peores (la diversificación será mayor) y cuando la temperatura vaya descendiendo, será menos probable aceptar soluciones peores y por tanto la intensificación será mayor.

* La fórmula que determina la **probabilidad** de **aceptar** una solución vecina es la siguiente:

$$P(\Delta E)=\begin{cases}1 & \text{if } \Delta E<0\\ \exp(-\Delta E/kT) & \text{otherwise}\end{cases}$$

- **ΔE**: Es la diferencia de fitness entre una solución y su vecina.
- **k**: Constante de Boltzmann (usada en termodinámica, en nuestro caso, es un valor que nosotros definimos, concretamente 1)
- **T**: Temperatura

* Para enfriar hemos usado la estrategia **exponencial**:

$$T=\alpha\cdot T$$

- **α**: Probabilidad (En nuestro caso α = 0.99)

* La temperatura inicial viene determinada por:

$$T=\frac{-\Delta E}{\log(\alpha)*k}$$

- **α**: Probabilidad (En nuestro caso α = 0.99)

Para ambos problemas hemos implementado 3 funciones diferentes. Una función para calcular la temperatura inicial, otra para otra para enfriar y otra que finalmente acepte o no la solución:

```
double TemperaturaInicial()
bool Aceptar(Solucion &s1, Solucion &s2, double &T)
double Enfriar(const double &T,const int &it)
```

Para la ***temperatura inicial*** se han generado 10 soluciones aleatorias y 10 soluciones vecinas a éstas. Para cada pareja de soluciones hemos calculado el correspondiente **ΔE** y se ha realizado la media aritmética para determinar el **ΔE** de la fórmula de la temperatura incial.

Problema de la Mochila (KP) con Enfriamiento Simulado.

Para el problema de la mochila se ha utilizado un ***criterio de aceptación*** que consiste en calcular el **ΔE** y si este es **mayor que 0**, se **acepta** la solución puesto que ésta ha mejorado. Si el **incremento** es **negativo**, entonces **aplicamos la fórmula** que determina la probabilidad de aceptarla. Todas las soluciones generadas no sobrepasan la capacidad de la mochila.

Problema del Viajante de Comercio (TSP) con Enfriamiento Simulado.

Para el problema del viajante se ha utilizado el mismo criterio pero considerando que la solución **mejora** cuando **ΔE** sea **menor que 0**. En **caso contrario**, se aplica la **fórmula** al igual que en el caso anterior salvo que al **ΔE** se le **cambia el signo** para minimizar.

Greedy Iterativo (Iterated Greedy)

```
std::vector<double> IteratedGreedy(int num)
```

En primera instancia el algoritmo genera una primera solución usando heurística y emplea una búsqueda local empleando el algoritmo de primera mejora para optimizarla. Seguidamente entra en un ciclo donde se destruye parcialmente la solución, se reconstruye para posteriormente mejorarla con una búsqueda local y finalmente se selecciona una solución siguiendo un criterio determinado. Para la implementación de este método se han creado tres funciones diferentes y cuyas cabeceras son:

```
void greedy(const int &numElementos)

Solucion Destruir(const int &porcentajeDestruccion, std::vector< int > &eliminados)

void reconstruir(Solucion &parcial, const std::vector<int> &eliminados)
```

Problema de la Mochila (KP) con Greedy Iterativo.

Para el problema de la mochila se ha considerado como ***criterio*** para ***generar una primera solución*** con heurística que ésta no sobrepase la capacidad máxima de la mochila y que a su vez genere una serie de candidatos aleatorios ordenados de mejor a peor ratio. Se generan nuevos candidatos mientras uno consiga entrar en la solución. La fórmula para calcular el mejor ratio es la siguiente:

$$\frac{beneficioMaterial}{pesoMaterial}$$

Para ***destruir*** de forma parcial la solución actual se han considerado tan sólo los materiales que ya han entrado. Se genera una lista con las posiciones que contienen un 1 y se desordena de manera aleatoria eliminando una parte de la solución.

Para ***reconstruir*** se ha seguido el mismo criterio que el empleado para generar una primera solución de manera que partiendo de una solución destruida se pueda generar una nueva empleando heurística y complementándola con una búsqueda local de primera mejora.

Como ***criterio de aceptación*** se ha considerado aquella solución que mejora la anterior y que no supera la capacidad máxima. Si alguna supera esta capacidad es penalizada fijando el fitness como la resta de el valor de ésta y el peso de la solución actual.

Problema del Viajante de Comercio (TSP) con Greedy Iterativo.

Para el problema del viajante de comercio se ha considerado como ***criterio*** para ***generar una primera solución*** un algoritmo voraz aleatorio que selecciona 5 candidatos de forma aleatoria de entre todos los posibles y escoge el mejor de éstos. Este proceso se repite hasta agotar el número de nodos.

Para ***destruir*** de forma parcial la solución actual se elimina una porción de los nodos que la componen de forma aleatoria y se guardan para su posterior reinserción.

Para ***reconstruir*** se introducen de forma aleatoria los nodos extraídos anteriormente. Cabe recalcar que este paso no se realiza de manera heurística porque la solución obtenida sería la misma que la anterior puesto que, la posición en que se reubicaría cada nodo al calcular la distancia mínima, seria la que ocupaba anteriormente.

Como ***criterio de aceptación*** se ha considerado aquella solución que mejora la anterior y que minimiza el camino entre los distintos nodos.

Capítulo 2 - Evaluación

Con el objetivo de comprobar qué algoritmo de los implementados consigue obtener soluciones más óptimas y su evolución a lo largo de las distintas evaluaciones se han ido recogiendo los diferentes datos en cada una de éstas.

Como ***criterio de experimentación*** se ha llevado a cabo la ejecución de 50 pruebas independientes para los casos más pequeños y 20 en los más grandes para no alargar el tiempo de estas últimas.

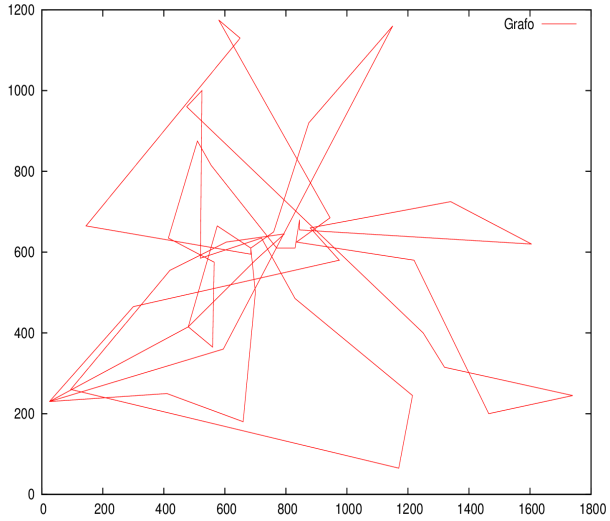
Para cada algoritmo, al observar que aproximadamente tras 1000 evaluaciones no se apreciaban cambios significativos y consultarlo con el profesor, se decidió fijar el número de éstas en dicho valor. Además de esto, no se ha considerado ningún otro criterio de parada.

A continuación, se mostrarán los resultados obtenidos en ambos problemas.

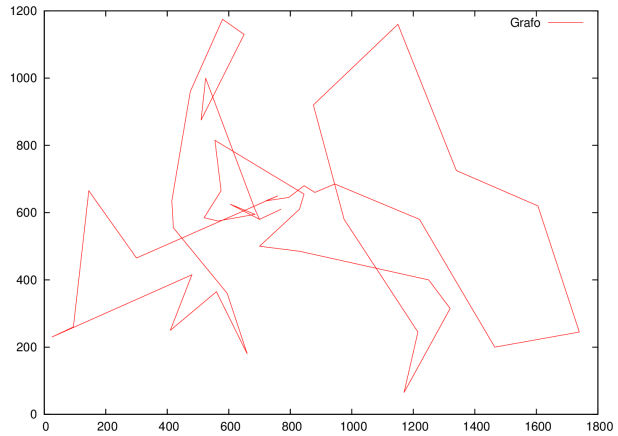
Grafos Viajante de Comercio

Para este problema se ha decidido, además de las gráficas de fitness frente iteraciones, representar la evolución del problema en grafos obtenidos en diferentes instantes de la evaluación.

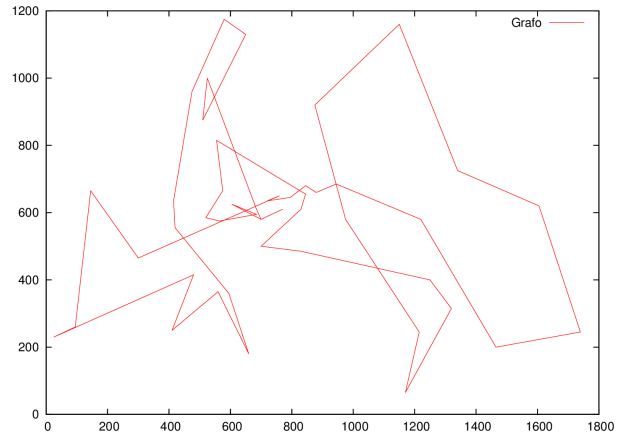
Evolución Greedy Iterativo Berlín_52



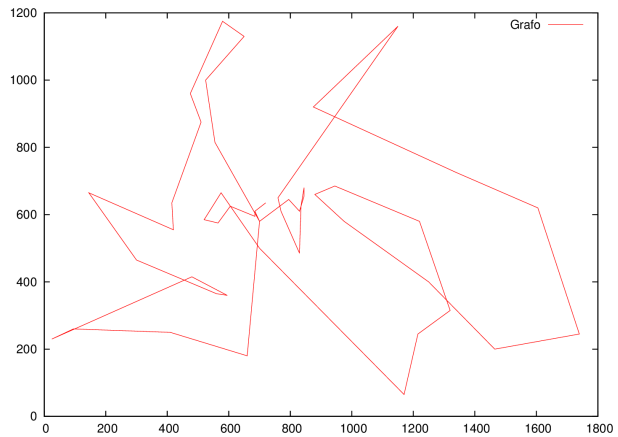
Grafo 1: Berlín_52 Iteración 0. Solución Inicial



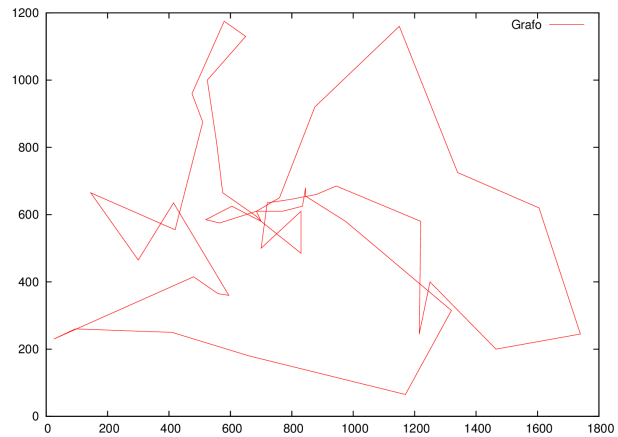
Grafo 2: Berlín_52 Iteración 0. Evaluación 250



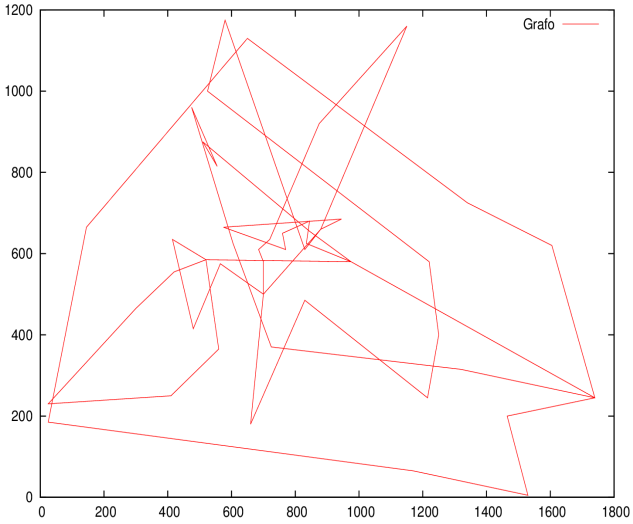
Grafo 3: Berlín_52 Iteración 0. Evaluación 500



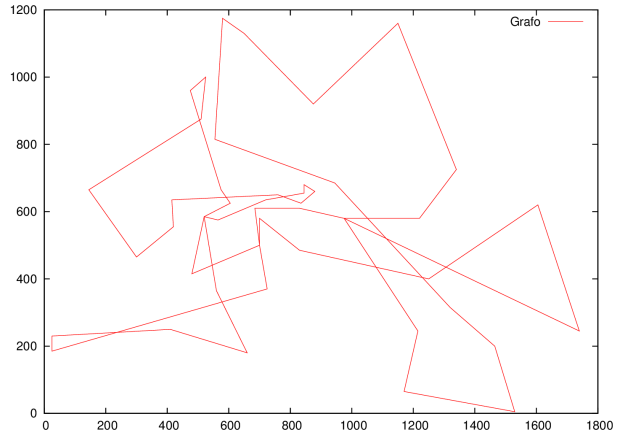
Grafo 4: Berlín_52 Iteración 0. Evaluación 750



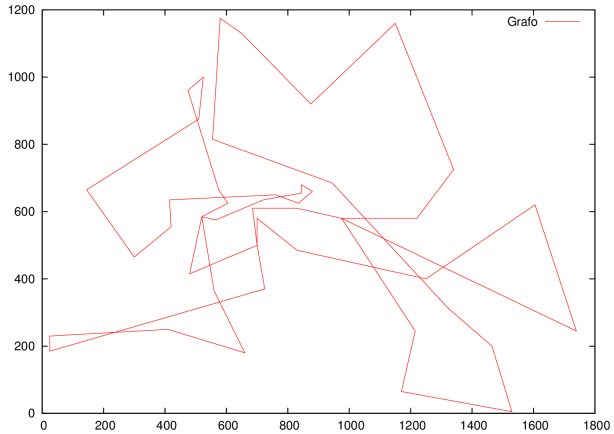
Grafo 5: Berlín_52 Iteración 0. Solución Óptima



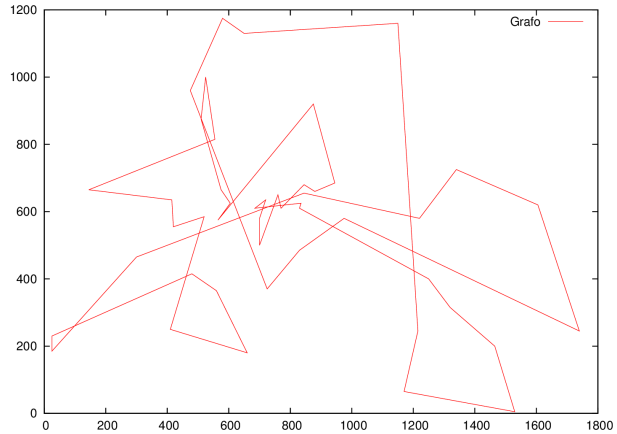
Grafo 6: Berlín_52 Iteración 48. Solución Inicial



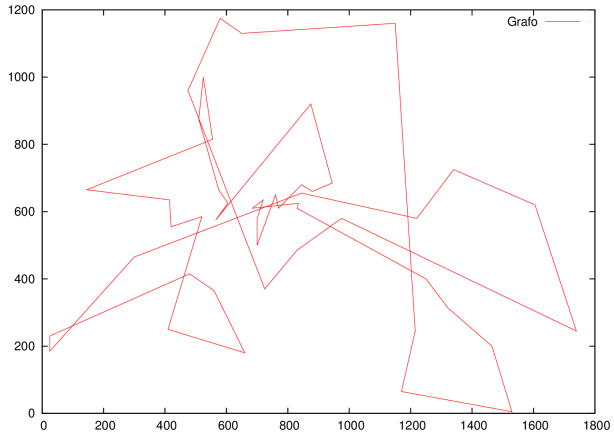
Grafo 7: Berlín_52 Iteración 48. Evaluación 250



Grafo 8: Berlín_52 Iteración 48. Evaluación 500

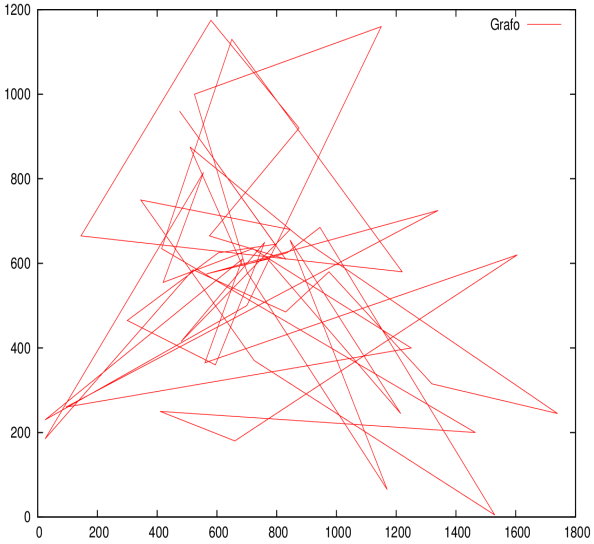


Grafo 9: Berlín_52 Iteración 48. Evaluación 750

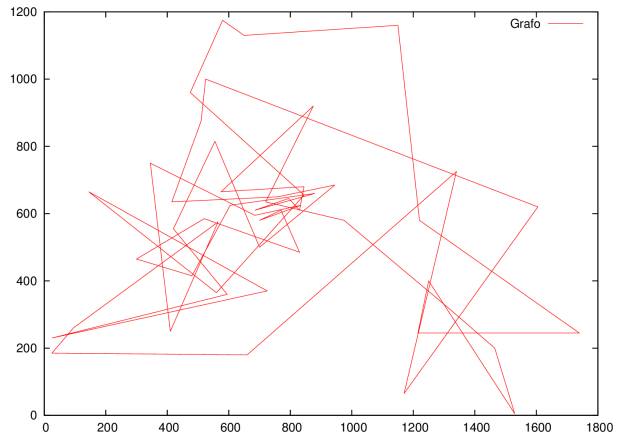


Grafo 10: Berlín_52 Iteración 48. Solución Óptima

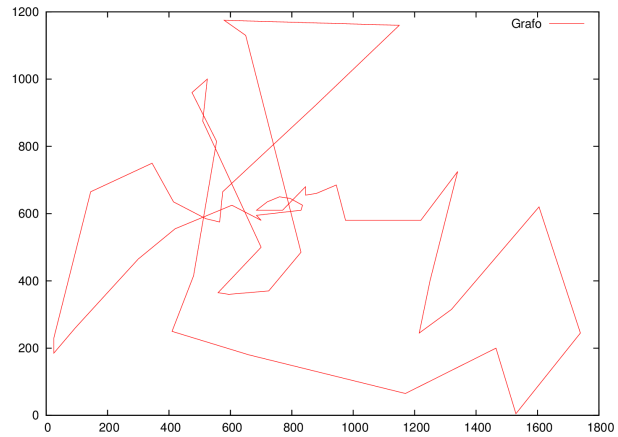
Evolución Enfriamiento Simulado Berlín_52



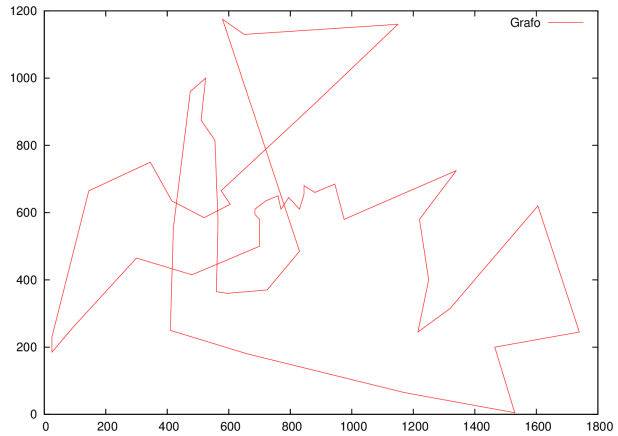
Grafo 11: Berlín_52 Iteración 32. Solución Inicial



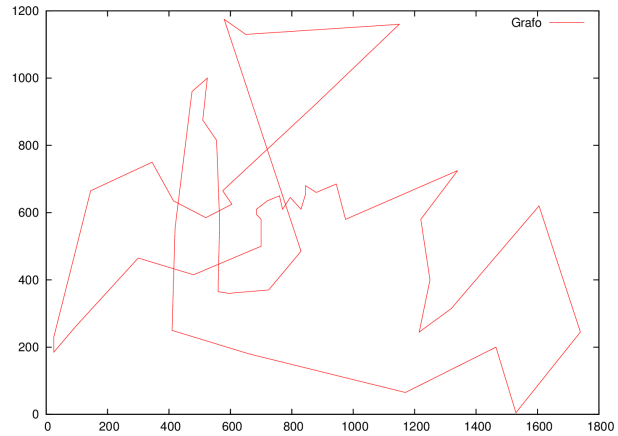
Grafo 12: Berlín_52 Iteración 32. Evaluación 250



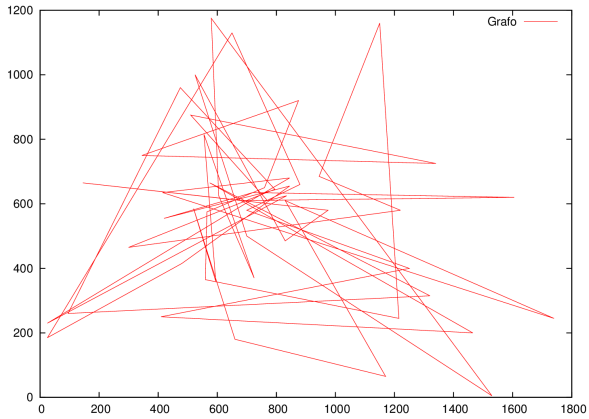
Grafo 13: Berlín_52 Iteración 32. Evaluación 500



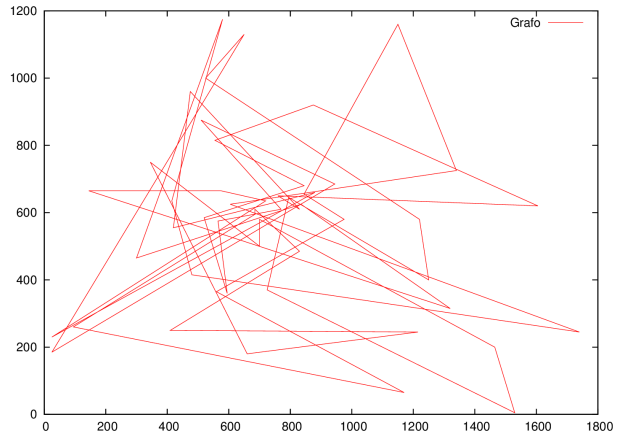
Grafo 14: Berlín_52 Iteración 32. Evaluación 750



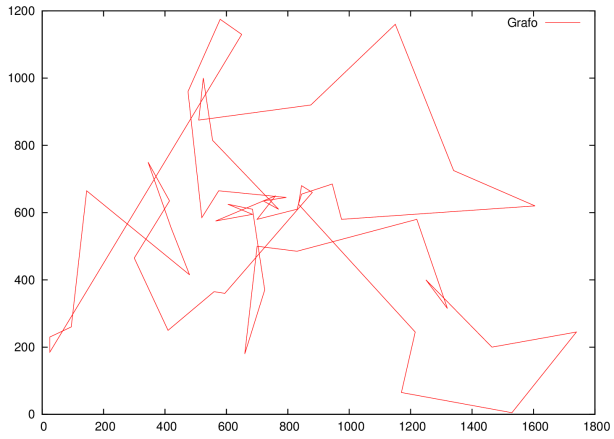
Grafo 15: Berlín_52 Iteración 32. Solución Óptima



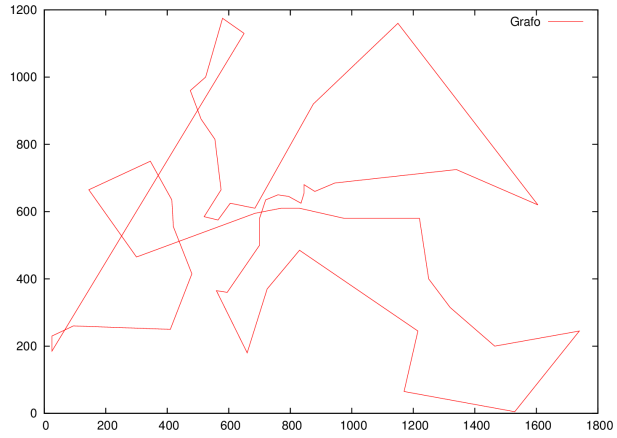
Grafo 16: Berlín_52 Iteración 48. Solución Inicial



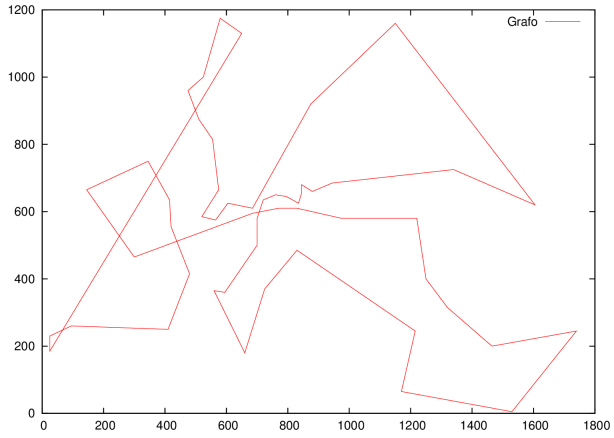
Grafo 17: Berlín_52 Iteración 48. Evaluación 250



Grafo 18: Berlín_52 Iteración 48. Evaluación 500

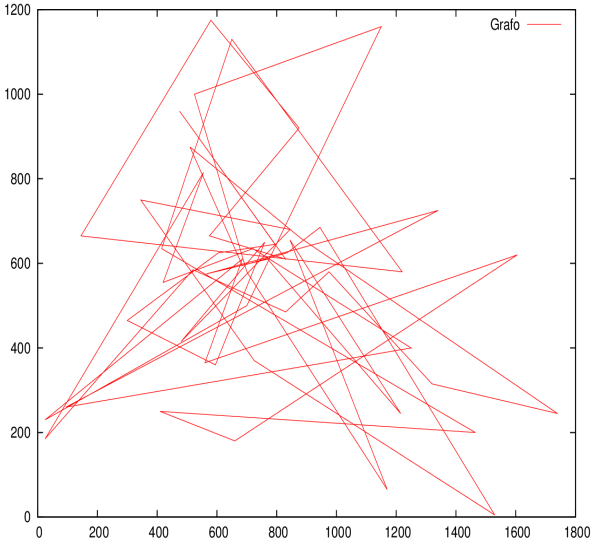


Grafo 19: Berlín_52 Iteración 48. Evaluación 750

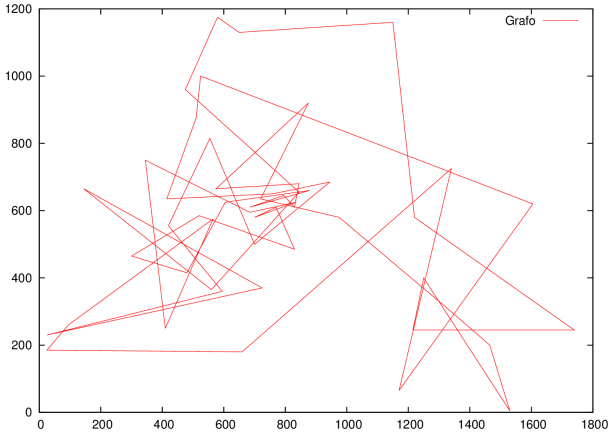


Grafo 20: Berlín_52 Iteración 48. Solución Óptima

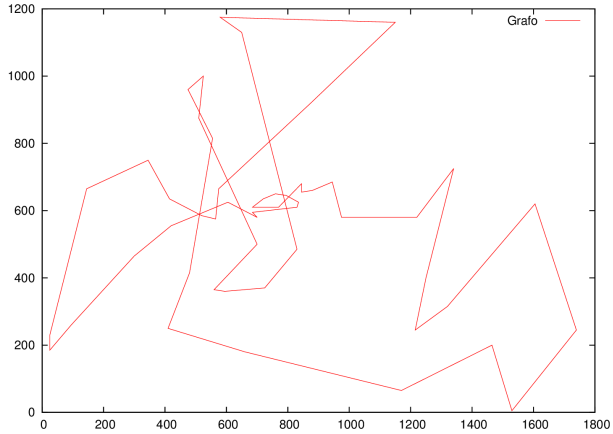
Evolución Enfriamiento Simulado Berlín_52



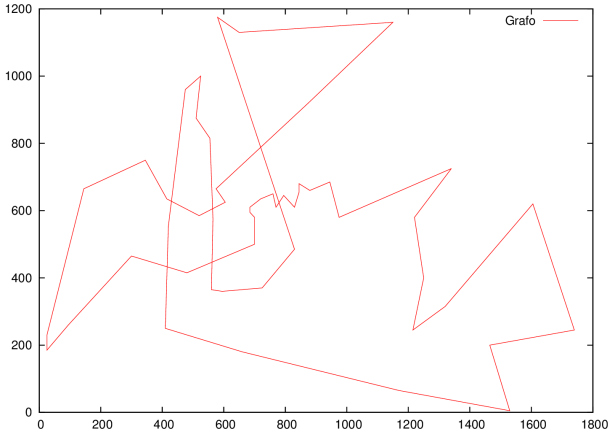
Grafo 21: Berlín_52 Iteración 32. Solución Inicial



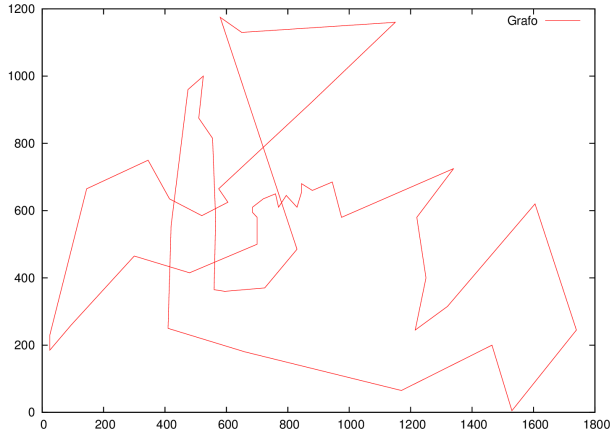
Grafo 22: Berlín_52 Iteración 32. Evaluación 250



Grafo 23: Berlín_52 Iteración 32. Evaluación 500

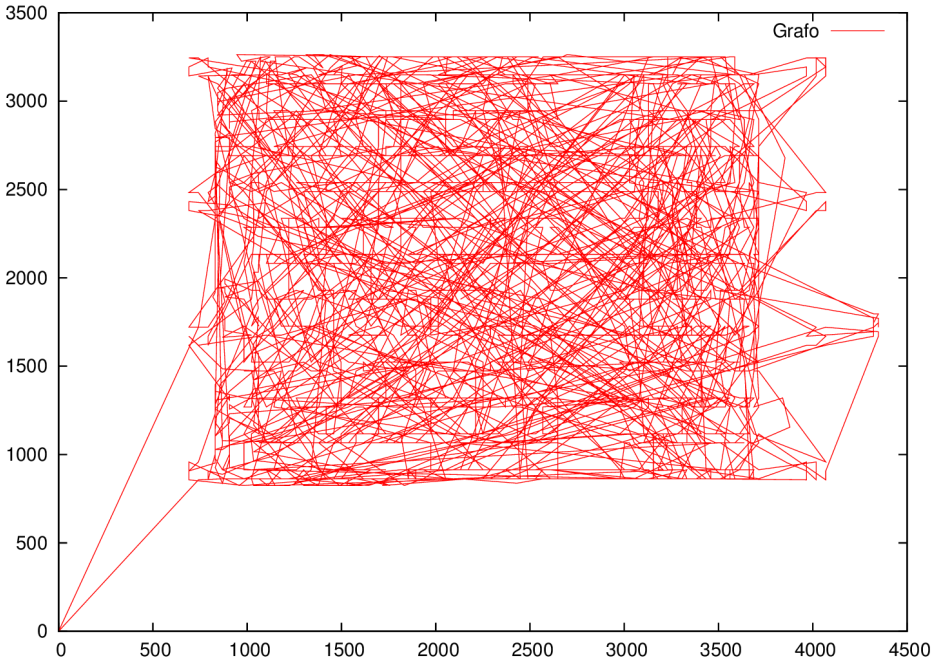


Grafo 24: Berlín_52 Iteración 32. Evaluación 750

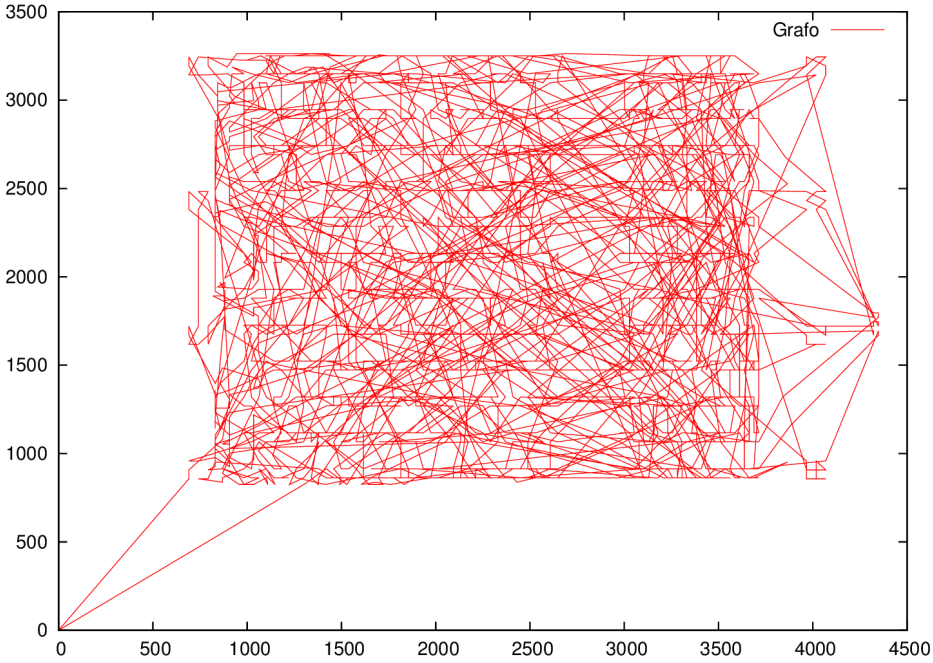


Grafo 25: Berlín_52 Iteración 32. Solución Óptima

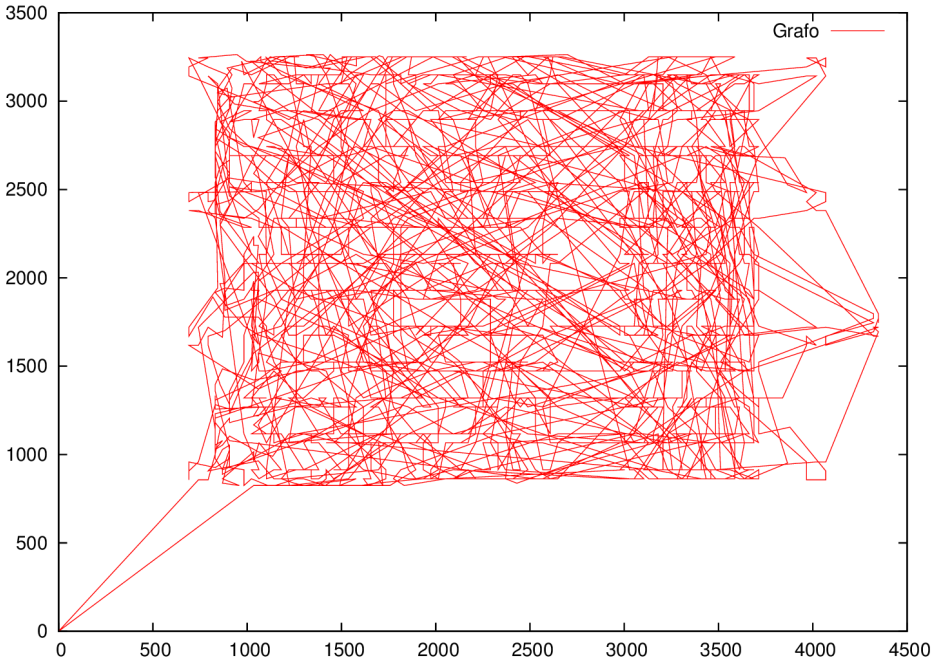
Evolución Greedy Iterativo d_2103



Grafo 26: d_2103 Iteración 0. Solución Inicial

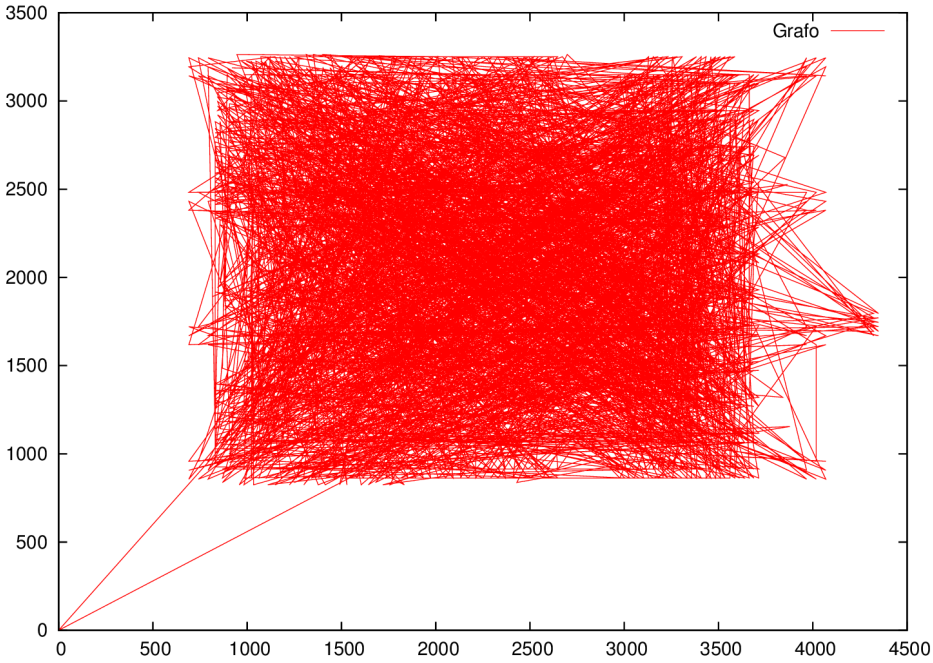


Grafo 27: d_2103 Iteración 0.Evaluación 500

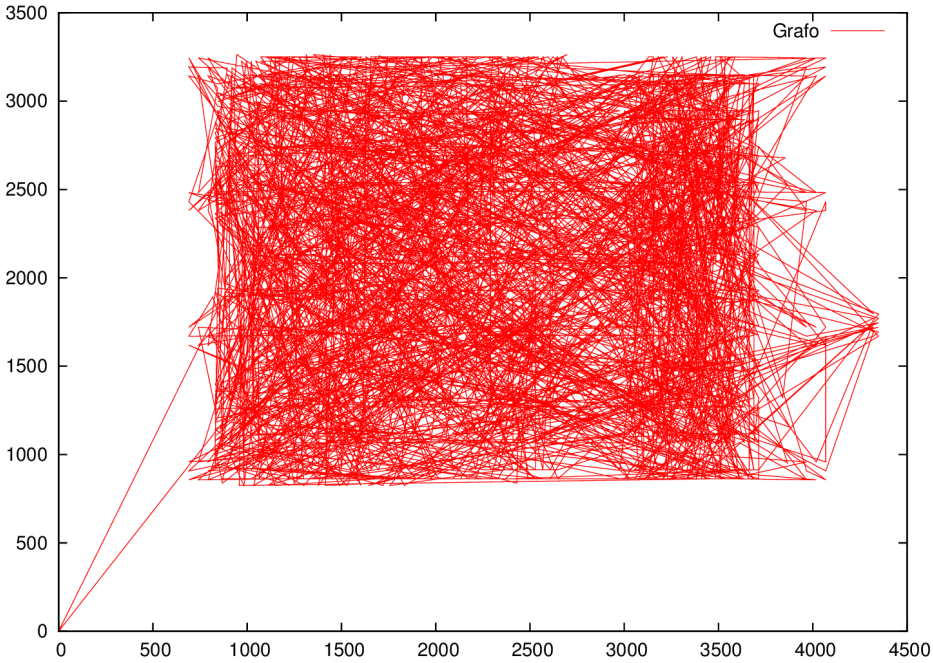


Grafo 28: d_2103 Iteración 0. Solución Óptima

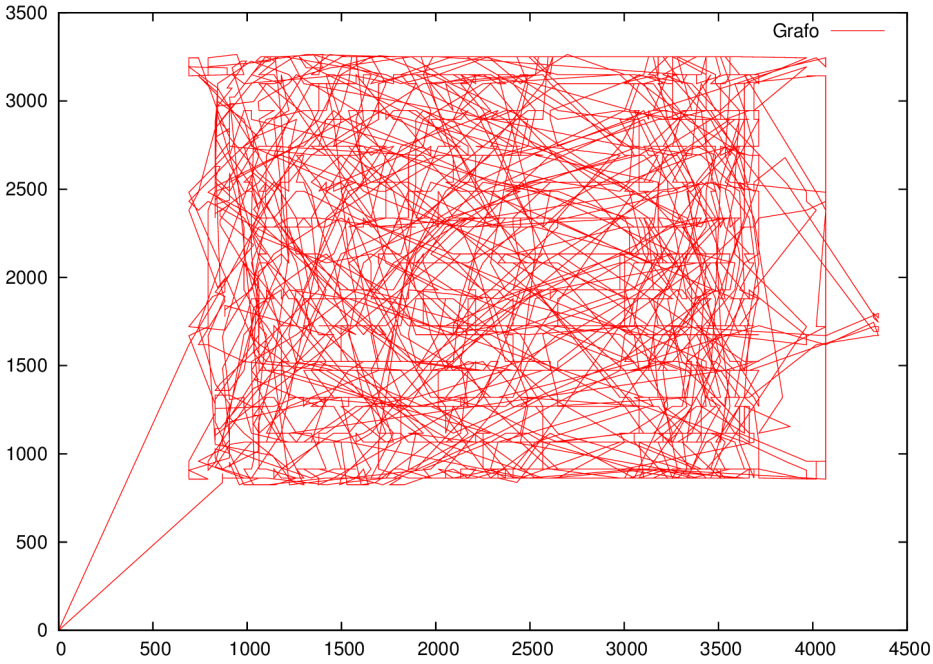
Evolución Enfriamiento Simulado D_2103



Grafo 29: d_2103 Iteración 0. Solución Inicial



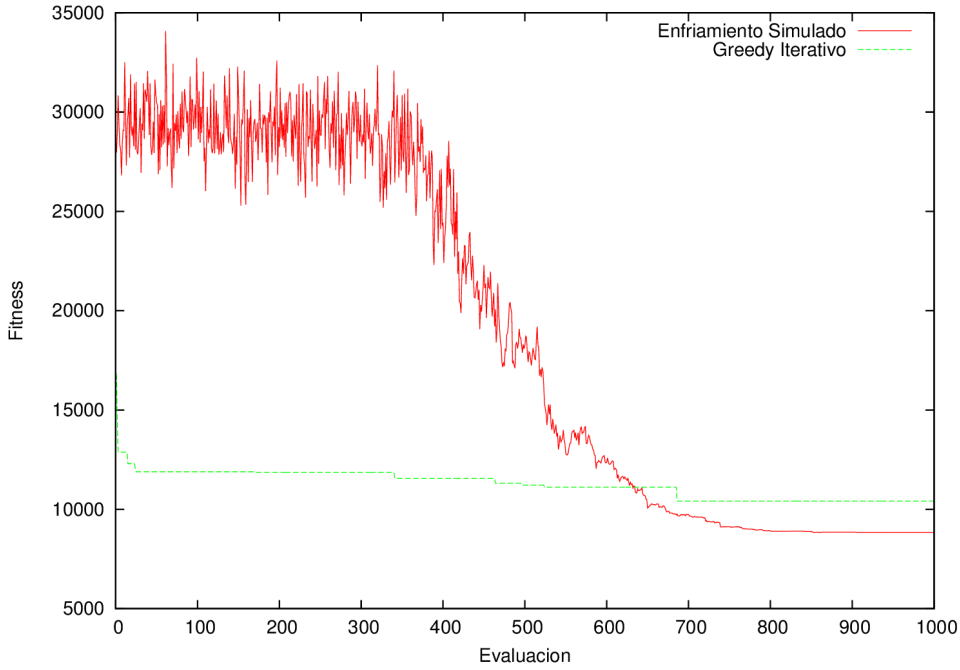
Grafo 30: d_2103 Iteración 0. Evaluación 500



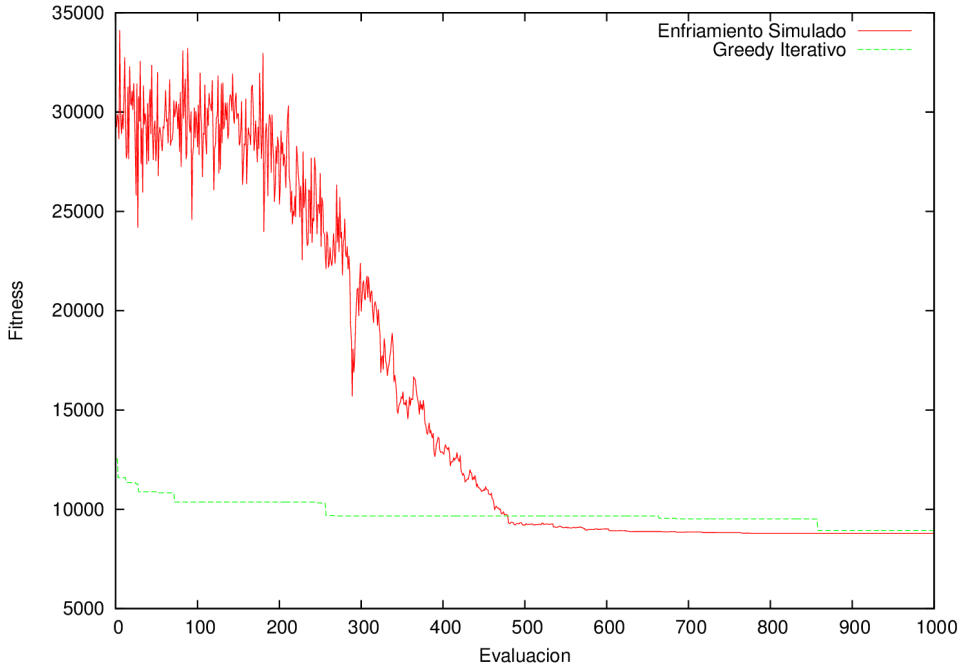
Grafo 31: d_2103 Iteración 0. Solución Óptima

Gráficas TSP

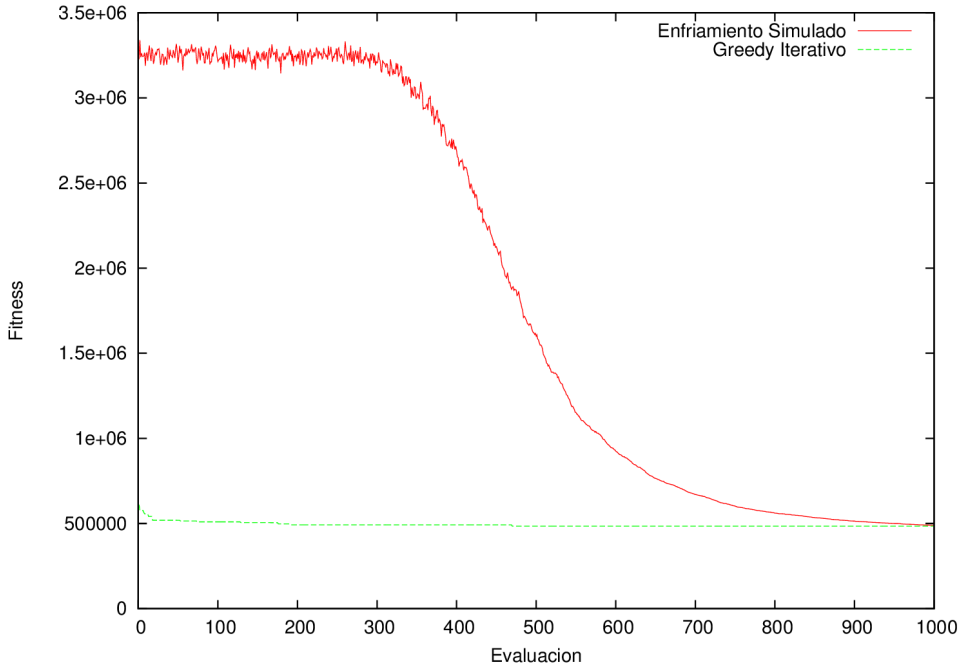
De las 50 gráficas generadas se han seleccionado las correspondientes a 2 iteraciones (las más interesantes observadas) de las totales para cada instancia. Además, se han seleccionado 2 instancias de las 3 disponibles (berlín_52 y d_2103) por considerarse más relevantes. A continuación se muestran:



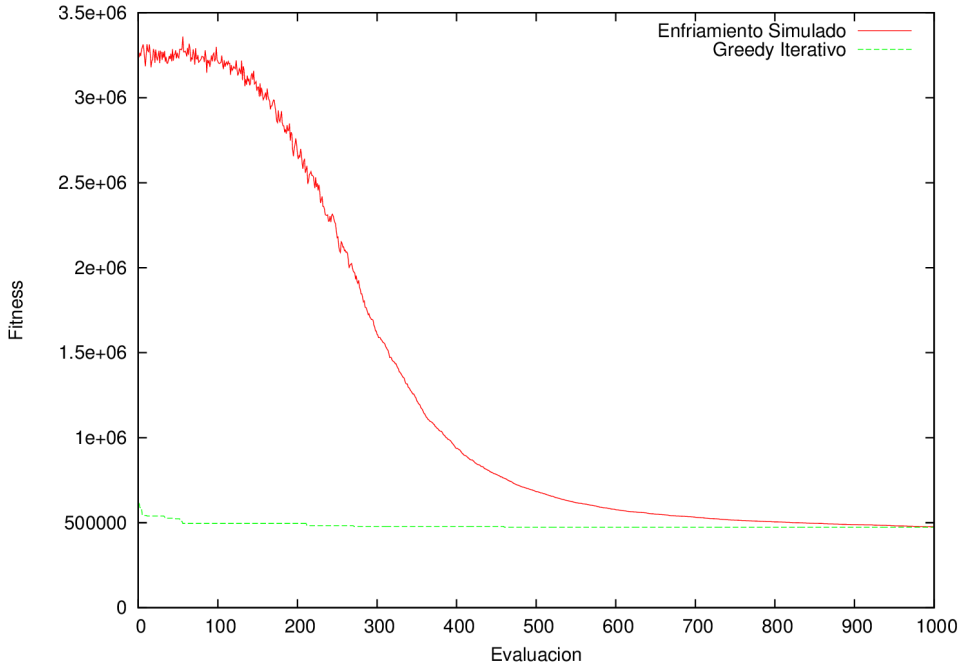
Gráfica 1: Berlín_52. Iteración 5



Gráfica 2: Berlín_52. Iteración 21



Gráfica 3: d_2103. Iteración 3



Gráfica 4: d_2103. Iteración 1

Conclusiones Gráficas TSP

Tabla de costes TSP Enfriamiento

- Berlín_52

9143,08	9712,8	9161,92	8517	9068,35	9480,25	8552,85	8836,82	9447,93	8848,01
9347,52	9011,36	8762,23	8308,61	9143,04	8713,17	9673	8645,97	9567,42	8222,24
9145,53	9575,32	8951,98	8891,16	8754,11	9399,25	9943,08	9417,19	8737,56	9333,05
8669,69	8503,86	8787,75	8806,97	9366,1	9064,91	9224,89	9115,56	8985,16	9613,39
8688,9	9219,05	8764,32	8931,9	8223,69	9392,16	8434,09	8774,93	8203,38	9107,49
Media:									9003,7998

- Ch150

Como se puede observar en las gráficas, los algoritmos evolucionan de la forma esperada:

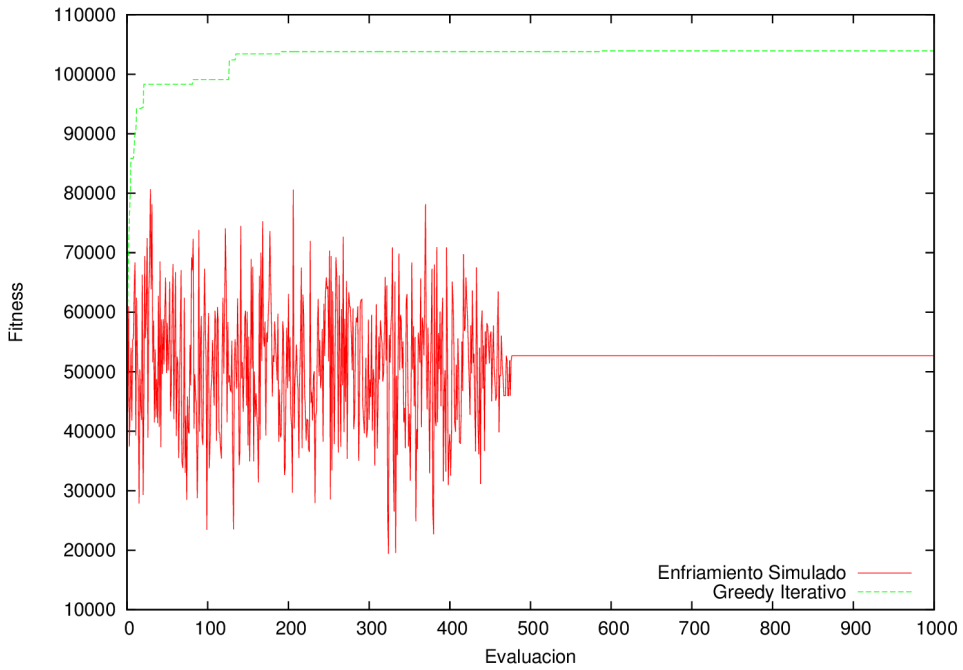
- En ***enfriamiento simulado***, debido a la alta temperatura en primer lugar se aprecia una diversificación porque acepta con mayor probabilidad soluciones peores. A medida que desciende la temperatura se intensifica la obtención de soluciones.
- En ***greedy iterativo***, se aprecia que la solución inicial se genera con heurística, concretamente con un greedy pseudoaleatorio, de modo que es mucho más óptima que la generada por enfriamiento. La linealidad de las soluciones obtenidas es debida a que en cada una de las distintas evaluaciones se aplica una búsqueda local para optimizar la generada con la destrucción y reconstrucción de la mejor solución anterior.

Sin embargo, al ir modificando los diferentes parámetros de ambos algoritmos se ha podido apreciar lo siguiente:

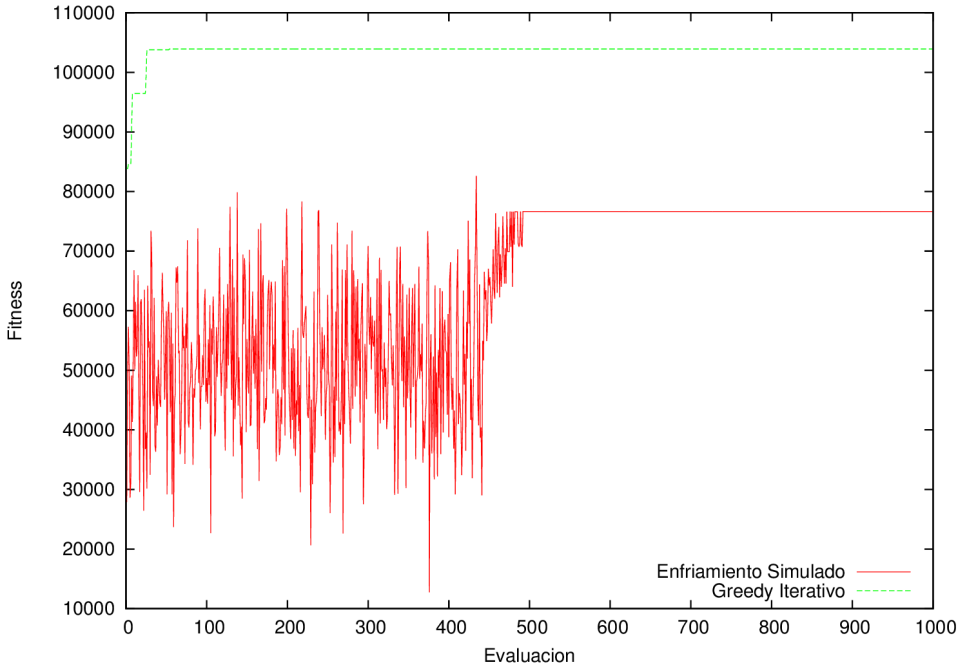
- En ***enfriamiento simulado***:
  - Al emplear la ***fórmula de Boltzmann*** para enfriar, comprobamos que la temperatura descendía con tanta rapidez que apenas se observaba la diversificación en las gráficas y por eso se ha decidido omitir dichos resultados.
  - En un principio, se usó como ***condición de parada*** la temperatura, pero era demasiado restrictivo y se decidió dejar como criterio de parada el número de evaluaciones para que diversificase mucho más.
  - Al disminuir la velocidad ($\alpha = 0.9999$) con la que desciende la temperatura el algoritmo busca soluciones más despacio (diversificando más) pero necesitaba un mayor número de evaluaciones para encontrar mejores soluciones. Finalmente, conseguía llegar a la misma solución que al enfriar un poco más rápido pero empleando un mayor número de evaluaciones.
- En ***greedy iterativo***:
  - Se ha podido comprobar que, al modificar el factor de destrucción de la solución, cuanto menos se destruya el vector de puntos mejores soluciones se obtienen en las distintas evaluaciones y viceversa. Esto es debido a que, al considerar el caso mas simple, todo los nodos están conectados entre si y esto hace que al destruir mucho dicha solución se obtiene una demasiado alejada de la actual.

Gráficas KP

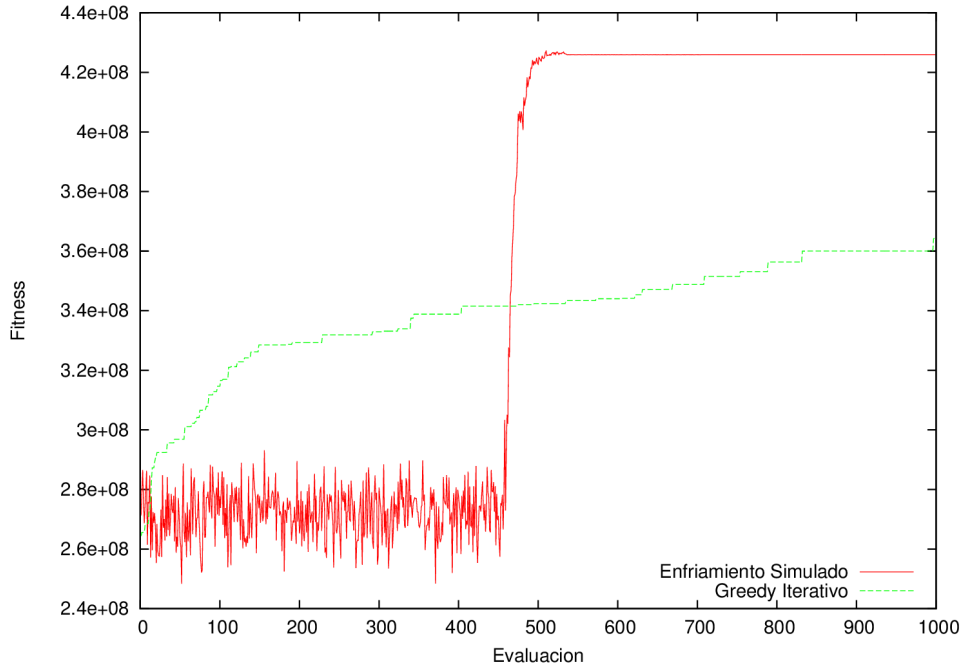
Análogamente a la sección anterior se mostrarán las dos gráficas más significativas correspondientes al problema de la mochila:



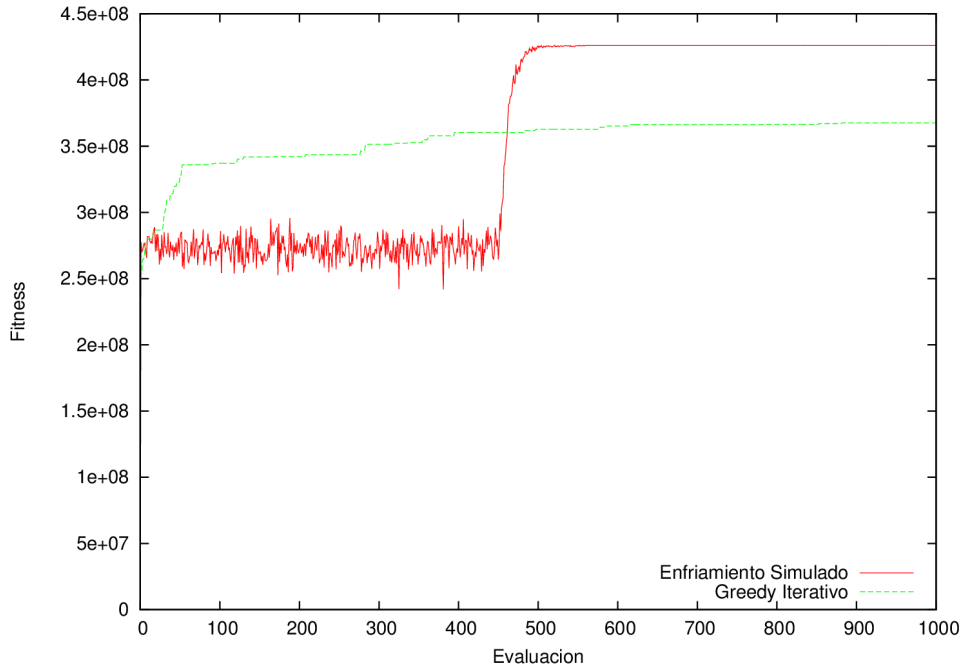
Gráfica 5: Knap_200. Iteración 10



Gráfica 6: Knap_200. Iteración 20



Gráfica 7: Knap_10000. Iteración 12



Gráfica 8: Knap_10000. Iteración 20

Conclusiones Gráficas KP.

Sadasdasd

Capítulo 3 - Conclusiones

A continuación presentamos las conclusiones extraídas del análisis de las gráficas vistas en el apartado anterior.

Problema del Viajante

En este problema concreto, se observa que al aumentar el número de nodos y vecinos, en primera mejora obtiene mejores resultados de tiempo y parecidos o poco mayores en coste, esto se debe a que la probabilidad de encontrar una primera mejor solución se incrementa reduciendo el tiempo necesario puesto que es una de las primeras encontradas. Calculando la probabilidad a priori de encontrar una solución que mejore la actual para primera mejora sería la probabilidad de elegir una solución que mejore (P) y la de que dicha solución se encuentre entre los k vecinos que hemos seleccionado.

$$\frac{P * k - vecinos}{vecinos}$$

Mientras que en Best la probabilidad de obtener un óptimo es uno porque recorre todo el vecindario hasta encontrar la solución mínimo coste de los k vecinos.

$$\frac{1}{vecinos}$$

Problema de la mochila

En TSP las soluciones **siempre** las forman **todos** los nodos, sin embargo, las soluciones en KP están condicionadas a los objetos que ya se encuentran en la mochila. Esto provoca que cuando tiene pocos materiales, Best se estanque en óptimos locales.

A pesar de que los tiempos obtenidos con First son mucho menores, la diferencia de beneficio en soluciones con muchos materiales (mucho mayor para Best) hace que sea más aconsejable usar Best en este tipo de problemas en los que el óptimo no contiene todos los posibles elementos del conjunto solución.

Conclusiones generales

En problemas en los que el óptimo incluye a todos los elementos del conjunto solución a priori es más aconsejable utilizar el método de la primera mejora aunque es posible que en algún caso no considerado esto no suceda, para asegurarse habría que realizar un experimento más amplio y con más instancias.

En problemas en los que el óptimo deseado está formando por un subconjunto del

conjunto solución parece más aconsejable usar mejor mejora ya que maximiza el beneficio para un coste asociado a este. Al igual que en el caso anterior, es posible que en algún caso no considerado ésto no se cumpla.

Teniendo en cuenta que primera mejora guarda el primer óptimo encontrado de forma aleatoria este método es mucho más diversificador que mejor mejora, mucho más intensificador que el anterior.

Capítulo 4 - Problema de la diversidad máxima.

Debido a la similitud del problema con el viajante de comercio a la hora de calcular los óptimos usaremos una de las clases implementadas en esta práctica. En esta clase crearemos un operador de vecindario que dada una solución inicial representada en una estructura de datos con los índices que identifican los diferentes puntos cambiará uno de dichos puntos por otro aleatorio diferente a los ya incluidos en la solución actual. El método de optimización que se aplique buscará minimizar la función de fitness con el objetivo de mejorar la solución.