Mejoras aplicadas al algoritmo de Simulated Annealing

En este documento explicaremos las mejoras que hemos aplicado sobre nuestro algoritmo original de Simulated Annealing así como los valores que usa el algoritmo final.

Mejoras aplicadas

- **Lista de soluciones candidatas**: En el algoritmo original simplemente se escogía el mejor vecino del vecindario de la solución actual. En dicho caso lo que ocurría muchas de las veces es que simplemente nos quedábamos iterando sobre las mismas soluciones una y otra vez. Para solucionar esto hemos hecho que se guarde en cada solución una lista enlazada de nodos ordenados según su coste. Luego, cada vez que preguntamos a la solución po el mejor vecino del vecindario esta nos devolverá la mejor solución una serie de veces, para luego pasar a devolvernos la segunda solución, luego la tercera, etc. Para evitar que se escojan selecciones muy malas se selecciona un número de soluciones sobre el cual volveremos a devolver la primera.
- **Reinicialización de soluciones**: También se ha observado que en el algoritmo original había un serio problema de falta de diversificación pues la mejor solución se obtenía en iteraciones tan tempranas como la segunda. Para ello hemos hecho que cada cierto número de iteraciones sin seleccionar la candidata como válida se haga un reseteo de la solución actual con una reinicialización greedy balanceada con una matriz de frecuencias. De esta forma, cada vez que se selecciona una solución se agrega a la matriz de frecuencias cada par de ciudades contiguas para que al reinicializar se escoja una solución de forma voraz con los pesos contrapesados con la matriz de frecuencias. Es decir, para construir una nueva solución inicial será menos probable que se introduzcan en ella soluciones que contengan transiciones entre ciudades muy frecuentes en soluciones candidatas.
- **Cambio de función de enfriamiento**: También se observó que la función de enfriamiento previa hacía que al principio se seleccionaran muchas soluciones y en un punto de la ejecución se pasara muy rápido a no seleccionar ninguna nunca más con lo que no se aprovechaban las 10000 iteaciones que le dábamos. Para ello hemos selecionado el Criterio de Boltzmann: $Tk = T0 / (1 + \log(k))$. Sin embargo si nos interesa la disminución rápida de la temperatura al principio pues nos permite empezar con temperaturas muy altas. A raíz de esto hemos implementado que se use la función original de enfriamiento hasta que el valor llegue a un límite para que luego se pase a usar el criterio de Boltzmann.
- **Cambio de los valores del algoritmo**:Además de los cambios anteriores también hemos realizado modificaciones sobre los valores que usa el algoritmo (MU, PHI, vecinos generados para enfriar y vecinos seleccionados para enfriar).

Valores utilizados

- MU: 0.50
- PHI: 0.008
- Iteraciones del problema: 10000
- Vecinos seleccionados para enfriar: 20
- Limite de temperatura para cambiar de cooldown: 2
- Vecinos generados para enfriar: 50
- Iteraciones sin seleccionar para reinializar: 60
- Peso de la matriz de frecuencias sobre los pesos reales: 0.005
- Numero de veces que se explora el mismo vecino: 50
- Maximo número de vecinos que se devuelven varias veces cada uno (módulo): 100

NOTA: Con estas mejoras y algoritmos hemos visto que no se tiene por que obtener resultados tam buenos como con menos mejoras (como por elemplo, se obtienen mejores resultados con más reinicializaciones, y usando siempre el mejor vecino de cada solución). Lo que obtenemos con este algoritmo es una mejora mas o menos constante, siendo capaz de aprovechar mejor las iteraciones que le damos devolviendo una solución cada vez mejor cuantas más iteraciones le aportemos para trabajar.